Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de herramientas para el control remoto de una fuente de alimentación mediante Python

Autor: Diego López Morilla

Tutores: Mª José Madero Ayora y Juan Antonio Becerra González

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Departamento de Teoría de la Señal y Comunicaciones

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Desarrollo de herramientas para el control remoto de una fuente de alimentación mediante Python

Autor:

Diego López Morilla

Tutores:

Mª José Madero Ayora

Profesor Titular

Juan Antonio Becerra González

Profesor Sustituto Interino

Dpto. Teoría de la Señal y Comunicaciones
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Grado: Desarrollo de herramientas para el control remoto de una fuente de alimentación mediante Python

Autor: Diego López Morilla
Tutores: Mª José Madero Ayora y Juan Antonio Becerra González

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mi familia
A mis amigos

Agradecimientos

Este Trabajo Fin de Grado cierra una de las etapas más importantes e influyentes de mi vida, por eso me gustaría agradecer en primer lugar, a toda mi familia por haberme hecho llegar hasta aquí, ya que sin su apoyo incondicional y su confianza en mí, no estaría escribiendo estas líneas ahora mismo.

Por otro lado, agradecer también a mis compañeros y amigos que me han acompañado en estos cuatro años de carrera, ya que ha sido un honor compartir este periodo con ellos, tanto los buenos momentos como los malos.

Por último, agradecer a los maestros que han sabido transmitirme sus conocimientos y han intentado dar lo mejor de sí mismos para formarme como ingeniero.

Sin más que añadir, me reitero en agradecer a todas estas personas lo que me han aportado durante estos años ya que todos y cada uno de ellos han contribuido en mi aprendizaje.

Diego López Morilla
Sevilla, 2018

Resumen

En este proyecto se analiza el control remoto de la instrumentación haciendo uso del estándar VISA (*Virtual Instrument Software Architecture*). Se han escrito una serie de programas en lenguaje Python para realizar el control remoto de una fuente de alimentación HP 6622A disponible en el Laboratorio de Radiocomunicación haciendo uso de un bus ampliamente conocido en la industria de la adquisición de datos, GPIB.

Para finalizar, se han realizado unas pruebas experimentales con dicho equipo empleando los códigos desarrollados y se han analizado posteriormente los resultados obtenidos.

Abstract

In this project, the remote control of the instrumentation is analyzed using VISA (Virtual Instrument Software Architecture) standard. A set of programs has been written in Python to perform remote control of an HP 6622A power supply available in the Radiocommunication Laboratory.

Finally, experimental tests have been carried out with this equipment using the developed codes and the results obtained have been analyzed later.

Índice Abreviado

Índice

1 Introducción

El valor de un acto reside más en el esfuerzo por llevarlo a cabo que en el resultado.

Albert Einstein

Actualmente, los sistemas de monitorización y control se llevan a cabo a través de instrumentación virtual ya que el avance tecnológico que se está experimentando es notable. Los sistemas basados en instrumentación virtual son empleados debido a su gran flexibilidad y capacidad de reconfiguración, así como su bajo coste y alto rendimiento. Siguiendo la línea de resaltar las ventajas de dicho sistema, cabe destacar el ahorro del tiempo respecto a la instrumentación tradicional.

En este proyecto se elaborará en Python un programa que permita controlar la fuente de alimentación HP 6622A remotamente.

La caracterización experimental de sistemas de radiocomunicación requiere el uso de equipamiento especializado de laboratorio que incluye generadores y analizadores vectoriales de señal. Con frecuencia, la realización de una medida conlleva la repetición de gran cantidad de configuraciones o la adquisición de grandes cantidades de datos, por lo que es habitual realizar dichas medidas de forma automatizada mediante el control remoto de los equipos de instrumentación. Por otro lado, además de automatizar dichos instrumentos, los DUT (*Device Under Test*) pueden necesitar una alimentación la cual puede variar de una medida a otra, por lo que también es realmente interesante monitorizar las fuentes de alimentación.

La instrumentación actual de comunicaciones permite diversas opciones para su control remoto, entre las que destaca el uso del estándar VISA (*Virtual Instrument Software Architecture*). Por otro lado, Python es un lenguaje de programación interpretado, multiplataforma y de código abierto, que está ganando importancia en los últimos años. Python cuenta con muchos paquetes y módulos puestos a disposición de cualquier desarrollador para realizar distintas tareas, entre ellas el control remoto de instrumentación mediante VISA.

1.1 Objetivos y alcance

En este proyecto, principalmente se llevará a cabo lo previamente expuesto. Mediante el control remoto se controlará una fuente de alimentación particularizando en un lenguaje concreto de programación, en este caso Python. Para lograrlo se desarrollarán funciones las cuales pondrán ser empleadas, por ejemplo, para establecer una tensión y una intensidad determinada, limitar intensidades, etc.

Para ello, previamente se expondrá teóricamente todo el contenido necesario para la comprensión de lo que se va a realizar en el laboratorio. Se hará un breve repaso de los buses más empleados en la actualidad para la interconexión de instrumentos, haciendo especial énfasis en el bus GPIB, el cual será el que se utilizará en este trabajo. Por otro lado, también se hará un pequeño resumen de lo necesario acerca de Python, explicando también el módulo PyVISA, el cual será muy importante para la tarea que queremos realizar.

Por último para demostrar la validez de todo lo previamente expuesto, se realizarán múltiples pruebas en el laboratorio como la caracterización la curva I-V de un transistor o la comprobación de la estabilización de la intensidad en un montaje al dejar la tensión constante.

1.2 Organización y estructura de la memoria

El presente Trabajo Fin de Grado se desarrolla en gran medida en el Laboratorio de Radiocomunicación de la Universidad de Sevilla por lo que en éste se pueden diferenciar dos grandes partes. Por un lado la parte teórica, desde el Capítulo 2 hasta el Capítulo 6, en la que se hará una descripción de los conocimientos necesarios para abordar el tema a tratar y por otro la parte práctica que la componen el resto de Capítulos, desde el 7 al 8, en la que se expondrán los códigos y experimentos realizados en este trabajo. A partir del presente Capítulo 1 de Introducción, se encuentran los siguientes:

- **Capítulo 2: Instrumentación Virtual.** Se presenta la instrumentación y su evolución desde la tradicional a la virtual. Se describe el estándar VISA, presentando su importancia para la instrumentación virtual al lector.
- **Capítulo 3: Otros buses para control remoto de instrumentos.** Se introducen y se hace una breve descripción de los principales buses empleados en la actualidad para la realización de instrumentación virtual. En este capítulo no será explicado el bus empleado en este trabajo ya que se le dedicará un capítulo exclusivamente para poder realizar un análisis más detallado del mismo.
- **Capítulo 4: GPIB: General Purpose Interface Bus**. En este capítulo se analizarán las características más relevantes del bus y se explicarán los comandos que se emplearán en la práctica. Además de ello, se realizará una descripción de la evolución de los estándares desde su aparición hasta el último estándar desarrollado.
- **Capítulo 5: Equipo utilizado.** Será empleado para la presentación de los instrumentos usados en la parte práctica del proyecto. Se hará especial énfasis en la descripción del equipo principal de este proyecto, la fuente de alimentacion HP 6622A, la cual va a ser controlada remotamente.
- **Capítulo 6: Python.** Servirá para introducir el lenguaje usado en la parte práctica, Python. Mediante este capítulo simplemente se pretende dar una visión general acerca de los aspectos básicos necesarios para entender los códigos desarrollados y no hacer una guía de programación.
- **Capítulo 7: Códigos desarrollados.** En este capítulo se expondrán los códigos que se han redactado para realizar las pruebas experimentales. Estarán acompañados de una breve descripción para su correcta comprensión por parte del lector.
- **Capítulo 8: Pruebas experimentales.** Se llevarán a cabo múltiples pruebas en las que se podrán a prueba el funcionamiento de los códigos previamente desarrollados. Mediante estas pruebas, se analizará el comportamiento de distintos dispositivos tales como transistores o amplificadores.
- **Capítulo 9: Conclusiones y líneas futuras de trabajo.** Se exponen las conclusiones a las que se han llegado tras la elaboración de este trabajo y se proponen actividades a realizar en el futuro en base al mismo.

2 Instrumentación Virtual

La potencia está en el Software.

NATIONAL INSTRUMENTS

La instrumentación virtual aparece a partir de la necesidad de emplear el ordenador personal como instrumento de medida de señales como por ejemplo: temperatura, velocidad, revoluciones, etc. Es decir, el PC recibe la información digitalizada procedente de un dispositivo externo, el cual realiza mediciones de fenómenos físicos y las convierte en señales de corriente o voltaje. Por otro lado, el ordenador no solo adquiere las medidas, sino que también realiza las tareas de procesamiento, análisis, almacenamiento, distribución y despliegue de los datos obtenidos. Esta es la base de los instrumentos virtuales, los cuales tienen sus funciones definidas por software, esto es lo que se conoce como instrumentación virtual. En este contexto destaca LabVIEW (*Laboratory Virtual Instrument Engineering Workbench*), el cual es el primer software utilizado para diseñar instrumentos en el ordenador empleando programación gráfica, esto es, que los programas no se escriben, sino que se dibujan, facilitando su comprensión.

En resumen, el instrumento virtual además de adquirir la señal, implica la interfaz hombre-máquina, el estudio y el tratamiento de la señal. Para realizar esta tarea será necesario realizar un montaje como el de la Figura 2.1 en el que aparecen un controlador, generalmente un PC (*Controller*) en el cual habrá un software de control remoto ejecutándose, los instrumentos de medición necesarios (*Instrument1*, *Instrument2*, *Instrument3*) que deberán tener capacidad de control remoto y se comunicarán con el controlador mediante una interfaz de comunicación física (*Communication Interface*). Por último será necesario un dispositivo bajo prueba (DUT), el cual se comunicará con el controlador, en los casos que sea necesario, mediante otra interfaz (*DUT Control Interface*).

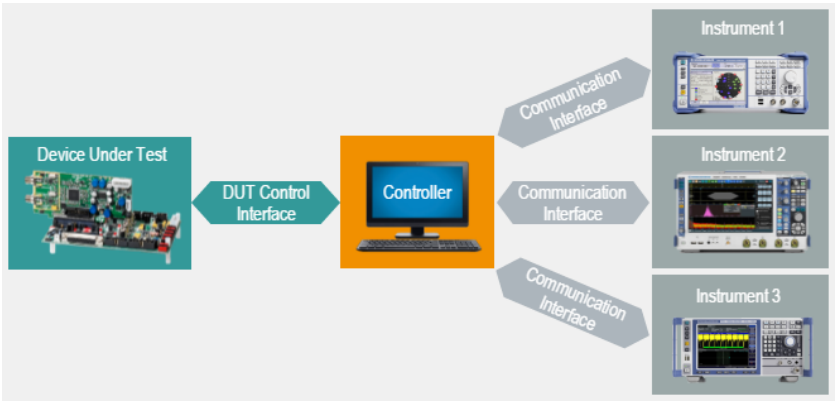


Figura 2.1 Esquema de montaje para automatizar medidas.

2.1 Evolución de la instrumentación

A partir de los últimos años, el uso de la tecnología está siendo uno de los factores mas decisivos para la mejora de los sistemas electrónicos basados en equipos de cómputo (estacionarios y móviles). Esto provoca el uso de nuevos sistemas de instrumentación a través del uso de nuevo hardware usando software cada vez mas novedoso y potente. La instrumentación virtual proporciona un nuevo enfoque a la instrumentación tradicional, añadiendo nuevos elementos y características mediante avances tecnológicos basados en innovaciones científicas. Históricamente, los sistemas de instrumentación tradicional se han basado en realizar mediciones individuales empleando sensores o transductores para obtener variables físicas y convertirlas en señales eléctricas para posteriormente, ser interpretadas y procesadas. Mediante esta metodología, para medir varias variables, serán necesarios diferentes aparatos físicos, los cuales pueden tener o no múltiples interfaces, dificultando la gestión de los datos obtenidos. Esta es una de las principales desventajas de la instrumentación tradicional, ya que no se tiene interacción con equipos de cómputo en tiempo real, al contrario que en la instrumentación virtual, en la cual se pueden automatizar las medidas.

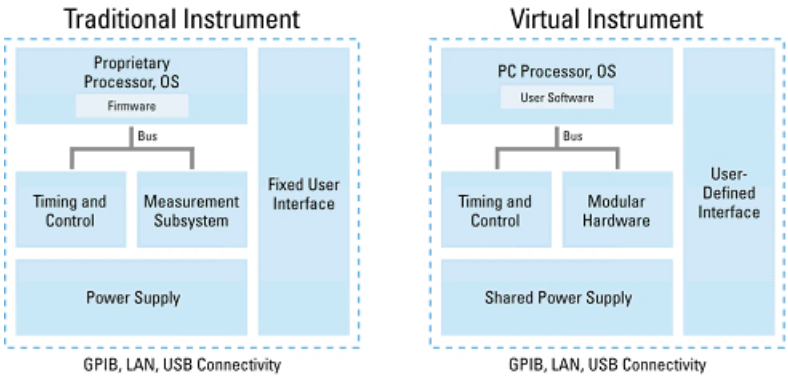


Figura 2.2 Instrumentación virtual vs. instrumentación tradicional.

2.1.1 Principales ventajas de la automatización de medidas

La idea principal de la instrumentación virtual, como ya se ha nombrado, es sustituir y ampliar elementos hardware por otros software, empleando para ello un ordenador personal. Esto nos va a proporcionar infinidad de ventajas con respecto a la instrumentación tradicional, ya que existirá un avance continuo en cuanto a flexibilidad y escalabilidad de equipos e instrumentos de medición. Las principales ventajas de automatizar las medidas son:

1. **Ahorro de tiempo**
   Algunas de las pruebas realizadas requieren de medidas repetitivas, ya que es necesario realizarlas para un amplio rango de valores. Empleando software, se podrá hacer esta tarea más rápidamente, siendo innecesario configurar los instrumentos para cada medida, incluso sin ni siquiera estar presente físicamente.
2. **Distancia de operación**
   Tal y como se ha nombrado previamente, no es necesario estar presente físicamente para realizar una medida, y ésta es otra de las ventajas de la instrumentación virtual. Empleando esto inteligentemente, se llega a la conclusión de que se pueden realizar medidas del DUT en distintas cámaras, como por ejemplo anecoica o de temperatura, para ver cómo se comporta y hacer las medidas a distancia.
3. **Mediciones repetibles**
   El proceso realizado para elaborar las pruebas siempre será el mismo por lo que una vez desarrollado y depurado, se podrá repetir en infinidad de ocasiones, lo que deriva en una mayor fiabilidad en la prueba realizada y los resultados obtenidos.
4. **Facilidad de expansión**
   Si el usuario desea realizar la misma tarea en más de un DUT, puede llevarlo a cabo de una forma más sencilla. Por ejemplo, un osciloscopio de cuatro canales puede ayudar a manejar cuatro DUT. Si se necesita algún DUT adicional, se puede usar una unidad de conmutación automática.

2.2 Estándar VISA

El estándar VISA (acrónimo de *Virtual Instrument Software Architecture*) es un API de alto nivel orientado al testeo y medida en la industria de la instrumentación virtual. Muchas de las empresas más reconocidas del sector implementan dicho estándar, como pueden ser Rohde & Schwarz, National Instruments, Keysight Technologies o Tektronix.

Mediante VISA, se puede configurar, programar y solucionar problemas de sistemas de instrumentación que comprenden interfaces GPIB, VXI, PXI, serie, Ethernet y/o USB ya que proporciona la interfaz de programación entre el hardware y los entornos de desarrollo como LabVIEW, Python, Visual Studio, LabWindows/CVI o Matlab. Tal como se muestra en la Figura 2.3, VISA permite al desarrollador programar independientemente del bus que vaya a ser empleado, ya que usando la misma librería se puede establecer la comunicación con distintos buses.

En resumen, VISA es un estándar de prueba y medida para permitir la comunicación entre un instrumento y una aplicación permitiendo a esta última acceder al instrumento sin necesidad de preocuparse de los detalles de la interfaz entrada/salida. Por lo tanto, para empezar a utilizar VISA, solo se tendrán que instalar las librerías VISA de algún proveedor y comenzar a llamar al archivo VISA DLL correspondiente. Un ejemplo de ello sería el caso en el que se usan comandos SCPI dentro de un lenguaje de programación y se comunica con un instrumento.

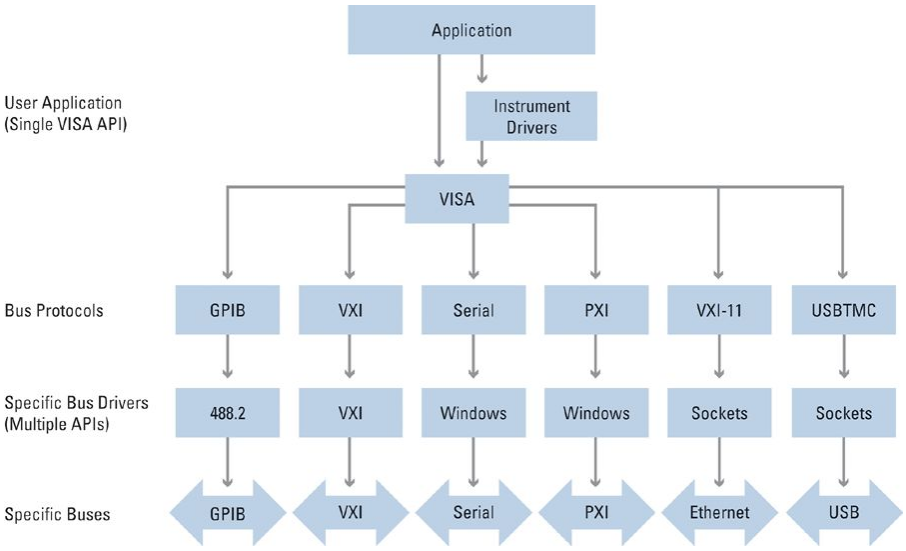


Figura 2.3 Empleo de VISA con distintos buses para la misma aplicación.

A continuación se expondrá una lista de las funciones VISA más importantes, las cuatro primeras hacen referencia a funciones de acceso y búsqueda y las cuatro últimas a funciones de entrada/salida.

VISA Open Default RM: Abre un nuevo administrador de recursos para todas las conexiones VISA.

VISA Find Resource / Find Next Resource: Encuentra un instrumento con unos criterios dados.

VISA Open/Close: Abre/cierra la conexión remota con un instrumento.

VISA Set/Get Attribute: Establece/lee un atributo.

VISA Write: Escribe un comando a un instrumento.

VISA Read: Lee la respuesta de un instrumento.

VISA Clear: Limpia el *buffer* de entrada y salida.

VISA Read STB: Lee el byte de estado de un instrumento.

2.2.1 Interfaces para el control de la instrumentación

Otra de las grandes ventajas de emplear VISA en instrumentación virtual es la capacidad de registrar la comunicación con los instrumentos. Existe un registro en el que se guardan todas las acciones realizadas, como pueden ser el envío de comandos o los mensajes recibidos por cada uno de los instrumentos conectados al bus con las marcas temporales correspondientes. Esta herramienta proporciona facilidad en la detección de errores en las comunicaciones y permite optimizar el rendimiento de las mismas. La interfaz de una de las herramientas disponibles de este tipo, VISA Trace de Rohde & Schwarz, se puede observar en la Figura 2.4.

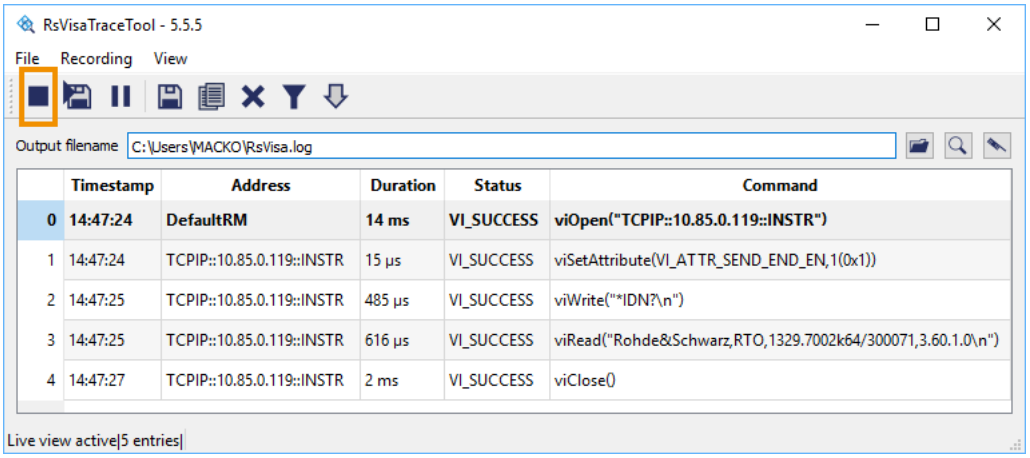


Figura 2.4 Interfaz gráfica de VISA Trace, ejemplo de herramienta para registrar la comunicación con instrumentos de Rohde & Schwarz.

3 Otros buses para el control remoto de instrumentos

Lo esencial es invisible a los ojos.

ANTOINE DE SAINT-EXUPÉRY

Durante más de 20 años, ingenieros y científicos han empleado el estándar IEEE 488, mejor conocido como GPIB, para automatizar sistemas de instrumentación. A medida que las tecnologías relacionadas con los ordenadores entran al contexto de la adquisición de datos y buses como USB y Ethernet son considerados para la conectividad de instrumentos, nacen nuevas alternativas y ponen en duda el futuro de GPIB como el bus más utilizado en cuanto al control de instrumentos. Debido a la gran cantidad de usuarios existentes de dicho bus y a su robustez, GPIB seguramente seguirá usándose por muchos años, sin embargo, la industria del control de instrumentos comenzará a abarcar un mundo de sistemas de entrada/salida (E/S o I/O).

Los sistemas de adquisición de datos pueden estar basados en instrumentos de adquisición independientes (Figura 3.1) o en sistemas modulares (Figura 3.2). El control de instrumentos independientes se puede llevar a cabo mediante un ordenador que permite la conexión de varios instrumentos a través de un bus de comunicación. Para sistemas modulares, la interconexión entre módulos se realiza a través de un bus de conexión local de altas prestaciones como el VXI o el PXI.

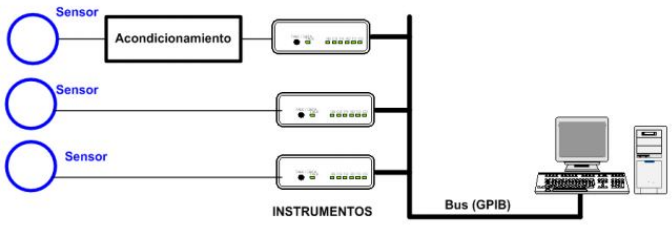


Figura 3.1 Sistema de adquisición de datos con instrumentos independientes.

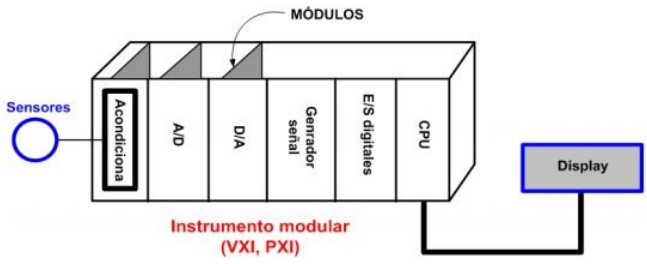


Figura 3.2 Sistema modular de adquisición de datos.

En este proyecto se hará uso de instrumentos independientes, por lo que se adoptará la primera de las dos topologías mencionadas, es decir, los instrumentos serán conectados a un ordenador a través de un bus de comunicación (GPIB). A continuación, en este capítulo se recogerán los protocolos más usados y extendidos en la actualidad.

3.1 USB: Universal Serial Bus

USB se basa en una arquitectura tipo serie, sin embargo, se trata de una interfaz I/O mucho más rápida que la de los puertos serie estándar, alcanzando velocidades de Gb/s. En el estándar que sigue, se definen cables, conectores y protocolos usados en el bus para comunicar las computadoras con los periféricos y dispositivos electrónicos.

3.1.1 Estándares USB

A partir de 1995, el estándar USB se ha ido desarrollando y evolucionando para mejorar la velocidad de transferencia:

- **Baja velocidad (USB 1.0)**
  Tasas de transferencia de hasta 1.5 Mb/s. Principalmente usado para teclados, ratones, etc.
- **Velocidad completa (USB 1.1)**
  Tasas de transferencia de hasta 12 Mb/s. Estos dispositivos dividen el ancho de banda de la conexión entre ellos empleando algoritmos LIFO.

Figura 3.3 Logotipo del USB 1.1.

- **Alta velocidad (USB 2.0)**
  Tasas de transferencia de hasta 480 Mb/s. El cable en este caso posee un par de línea para datos y otro par de alimentación.

Figura 3.4 Logotipo del USB 2.0.

- **Super-alta velocidad (USB 3.0)**
  Tasas de transferencia de hasta 4.8 Gb/s. La velocidad es mucho mayor debido a la inclusión de 5 contactos adicionales. Es compatible con los estándares anteriores.

Figura 3.5 Logotipo del USB 3.0.

- **Super-alta velocidad + (USB 3.1)**
  Alcanza tasas de hasta 10 Gb/s. El estándar USB 3.1 es compatible con USB 2.0 y 3.0, limitándose a la velocidad de estos. Esta especificación también es conocida como *SuperSpeed+*.

Figura 3.6 Logotipo del USB 3.1.

Además de la compatibilidad de USB 3.0 con las versiones anteriores, USB 1.0, 1.1 y 2.0 también lo son entre sí. A pesar de ello, el uso de un dispositivo USB 2.0 en un puerto de baja velocidad limitará su velocidad al máximo de la velocidad de dicho puerto.

3.1.2 Conectores

Las señales del USB se transmiten en un cable de par trenzado con impedancia característica de 90 Ω ± 15 %, cuyos hilos se denominan D+ y D-. Éstos, utilizan señalización diferencial en *half* dúplex, excepto el USB 3.0 que utiliza un segundo par de hilos para realizar una comunicación en *full* dúplex. La razón por la cual se realiza la comunicación en modo diferencial es que reduce el efecto del ruido electromagnético en enlaces largos. Los niveles de tensión para un nivel bajo son entre 0 V y 0.3 V y entre 2.8 V y 3.6 V para un nivel alto en las versiones 1.0 y 1.1, y en ±400 mV en alta velocidad (2.0).

USB contempla distintos tamaños y tipos de conectores con distintas especificaciones, los cuales pueden verse en la Figura 3.7. También se pueden observar en la Tabla 3.1 y en la Tabla 3.2 las conexiones de los pines de los conectores.

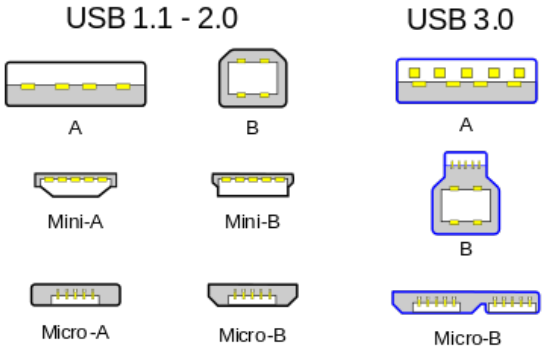


Figura 3.7 Tipos de conectores USB.

Tabla 3.1 Pines de los conectores USB tipo A y B.

Pin	Nombre	Color del cable	Descripción
1	VCC	Rojo	+5 V
2	D-	Blanco	Data -
3	D+	Verde	Data +
4	GND	Negro	Tierra

Tabla 3.2 Pines de los conectores USB tipo mini y micro.

Pin	Nombre	Color del cable	Descripción
1	VCC	Rojo	+5 V
2	D-	Blanco	Data -
3	D+	Verde	Data +
4	ID	-	Permite la distinción de Micro-A (conectado a tierra) y Micro-B (no conectado)
5	GND	Negro	Tierra

3.2 LXI: LAN eXtensions for Instrumentation

Se trata de un estándar desarrollado por el *LXI Consortium*[1] en el que se definen protocolos de comunicación para instrumentos que usan Ethernet. Dicho consorcio se fundó en 2004 y la versión 1.0 de LXI se lanzó en septiembre de 2005, actualizándose varias veces hasta el día de hoy. LXI es el estándar para los equipos de instrumentación de una red LAN que ayuda a reducir el tiempo de establecimiento, configuración y depuración de los sistemas bajo prueba. Es un estándar abierto y accesible basado en Ethernet que proporciona una solución para la medición y adquisición de datos. Sus principales beneficios son:

1. Reduce costes utilizando componentes de LAN.
2. Proporciona alto rendimiento.
3. Garantiza una amplia disponibilidad de instrumentos.
4. Posee un servidor web incorporado para acceso remoto y control del dispositivo.

Estos puntos anteriores son básicos de todos los dispositivos LXI. Además, hay una serie de dispositivos LXI que incluyen funcionalidad extendida, lo que posibilita que los dispositivos LXI se comuniquen entre sí usando *hardware triggers*, paquetes LAN o basados en la sincronización de tiempo.

3.2.1 Características principales

La versión más actual, la 1.4, está definida como un conjunto de funciones principales, las cuales son llamadas *Core* y otro conjunto de extensiones opcionales, como se puede ver en la Figura 3.8. A continuación se explicarán las características básicas del *Core*:

- **Estándares abiertos de la industria**
  LXI está basado en estándares muy utilizados en la industria, como pueden ser TCP/IP, IPv4/IPv6, navegadores web y controladores IVI.
- **I/O Ethernet de alta velocidad**
  LXI se basa en la tecnología Ethernet, la interfaz de comunicaciones más usada actualmente. Ethernet I/O proporciona compatibilidad con versiones anteriores y conexión estándar. Por otro lado, casi todos los ordenadores se fabrican con una interfaz Ethernet integrada, y el hardware de red se está volviendo cada vez más económico.
- **Interfaz web estandarizada incorporada**
  Con la interfaz web, los usuarios pueden, entre otras cosas, configurar y operar fácilmente los instrumentos con una GUI (*Graphical User Interface*), recopilar y analizar los datos sin programación software y operar los instrumentos de forma remota.

[1] LXI Consortium es una corporación sin ánimo de lucro desarrollada inicialmente por Keysight Technologies y VTI Instruments. Su principal propósito es promover el desarrollo del estándar LXI.

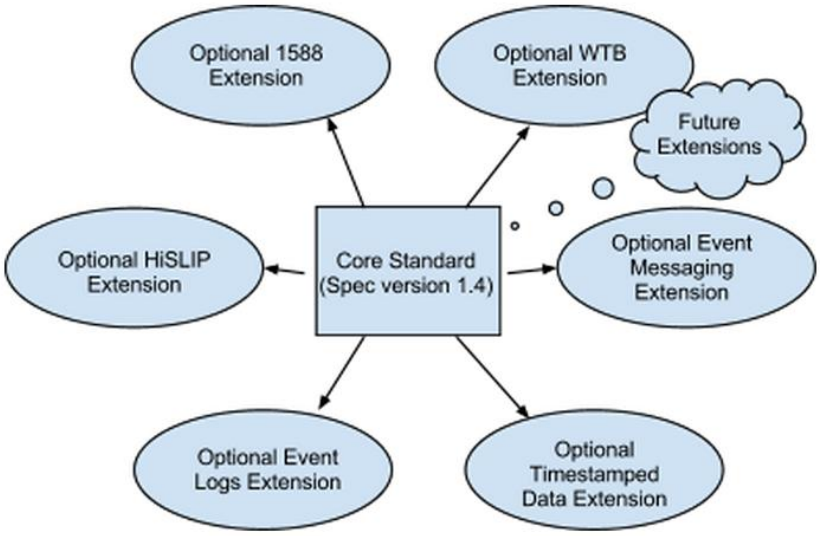


Figura 3.8 *Core* LXI Standard 1.4 con extensiones opcionales.

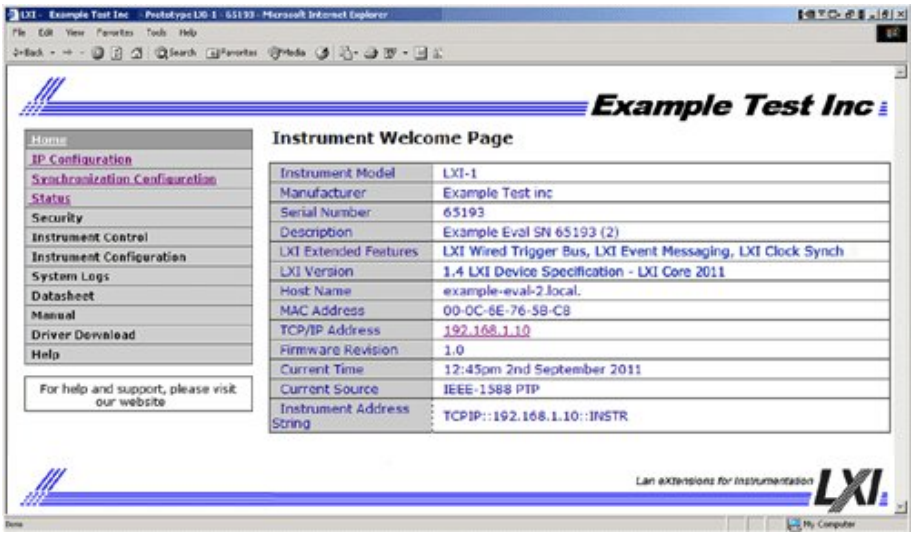


Figura 3.9 LXI GUI.

3.3 RS-232C: Recommended Standard 232

El puerto serie RS-232C es una de las formas más empleadas actualmente para la transmisión de datos entre dos equipos. Para minimizar el efecto del ruido electromagnético, los rangos de voltaje con los que se trabaja son: de 3 V a 15 V, que significa un 0 lógico, y de -3 V a -15 V, que significa un 1 lógico. Esto provoca que, por ejemplo, si se manda una señal a 5 V y llega algo modificada, se va a entender el 0 deseado perfectamente.

La versión europea se regula bajo la norma CCITT V.24 y especifica una distancia máxima de 15 m., además de una velocidad máxima de transmisión de 20 kb/s.

3.3.1 Conectores

Cuando se transmite a través de un módem la norma define un conjunto de 22 señales divididas en señales de datos y señales de control distribuidas en un conector tipo D de 25 terminales. A pesar de ello, algunas señales de control no son necesarias para establecer la comunicación entre dos equipos, es por eso que en muchas ocasiones se utiliza un conector macho tipo D de 9 terminales.

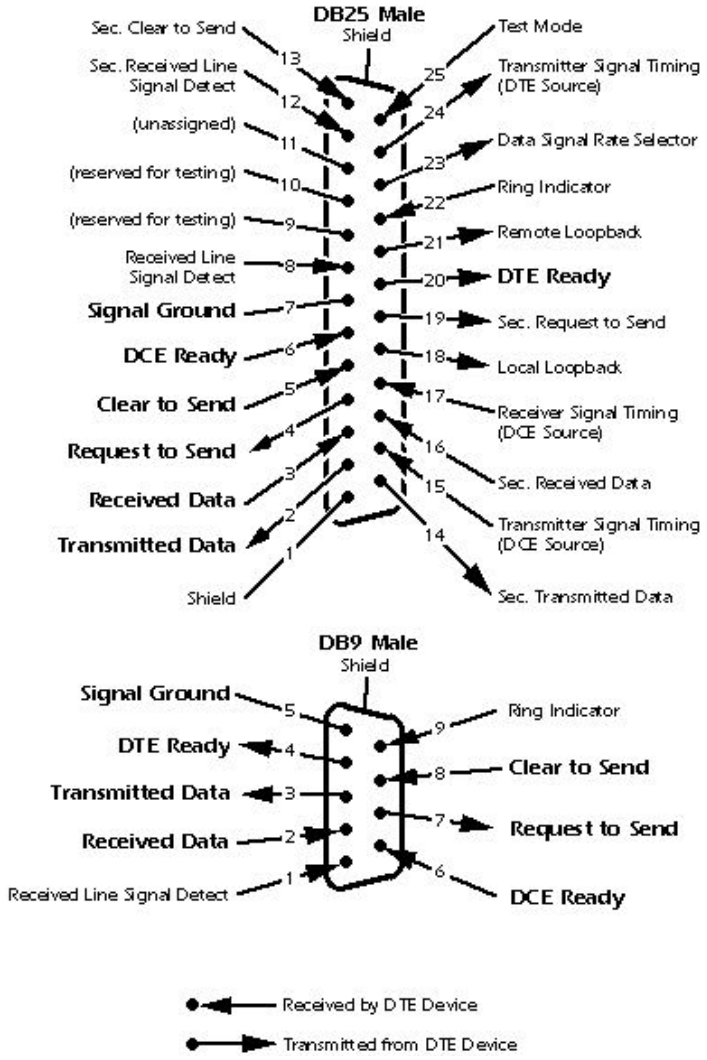


Figura 3.10 Pinout de los conectores DB.

Como se puede observar en la Figura 3.10, se puede establecer una comunicación con tan solo 9 señales, prescindiendo del resto, incluso podría reducirse a 3 líneas solamente empleando los pines de RX, TX y GND ya que el resto son necesarios en caso de querer implementar un mejor sincronismo entre dispositivos. Para realizar esto, una práctica muy común es emplear transceptores para adaptar los niveles TTL de los microcontroladores a los empleados en RS232 como se indica en la figura Figura 3.11.

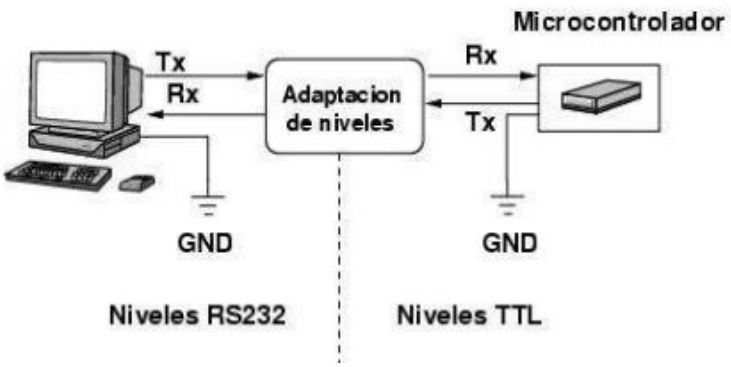


Figura 3.11 Conexión ordenador-microcontrolador usando un transceptor.

3.4 PCI Express: Peripheral Component Interconnect Express

Se trata de un estándar de comunicación para ordenadores basado en el bus PCI, el cual se utiliza conectando directamente periféricos a la placa base. A diferencia de los buses ISA, PCI permite una configuración dinámica del periférico, es decir, en el tiempo de arranque las tarjetas PCI y el BIOS se comunican para negociar los recursos solicitados por la primera. Se trata de un estándar fundamental ya que es utilizado para la comunicación de las tarjetas gráficas y puede emplearse para otros tipos de tarjeta como pueden ser de red, de sonido o de expansión. La versión más empleada en la actualidad es la 3.0, una mejora sobre la PCI 1.0 original la cual cuadriplica su velocidad de transferencia hasta 8 GT/s.

3.4.1 Conexiones

Las conexiones físicas de los puertos PCI-e pueden encontrarse hasta de cuatro tamaños distintos, como puede observarse en la Figura 3.12.

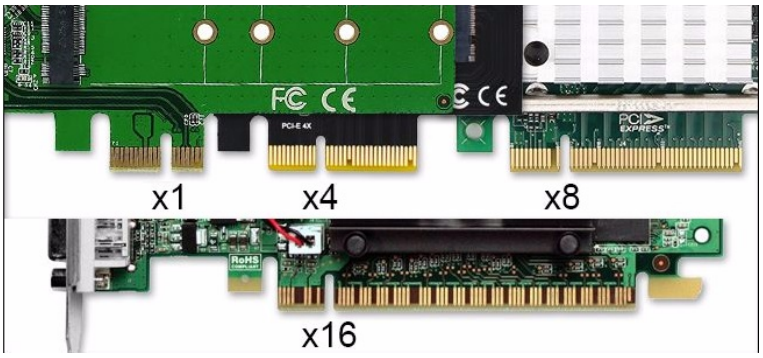


Figura 3.12 Ejemplos de conectores para PCI Express.

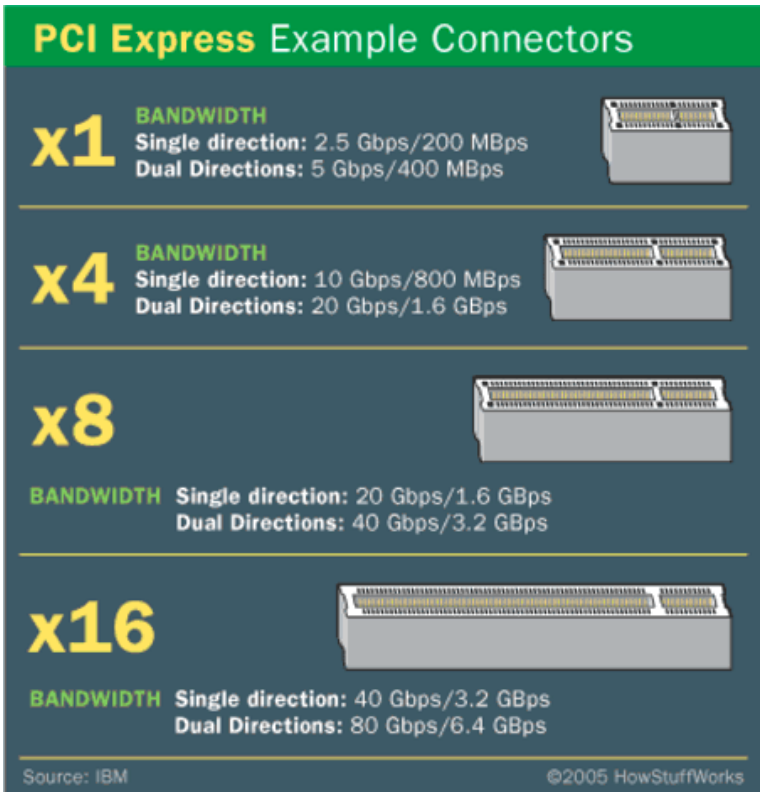


Figura 3.13 Conexiones físicas de los puertos PCI Express.

Mientras más grande sea el puerto, más conexiones y pines de datos tendrá. Estas conexiones se conocen como carriles, los cuales cada uno de ellos tienen dos señalizaciones, una diseñada para transmitir y otra para recibir. Por lo tanto, cuantos mas carriles existan, mayor será la velocidad a la que se transmitan los datos.

3.4.2 Versiones

Existen actualmente cuatro versiones y estamos a la espera de una quinta. En cada una de ellas la velocidad de transferencia por línea de datos ha sido mejorada. Simplificando mucho, tenemos unas velocidades de 250 MB/s en PCI 1.x, 500 MB/s en PCI 2.0, 1 GB/s en PCI 3.0 y 2 GB/s en PCI 4.0 por línea, todo esto se resume mejor en la Figura 3.14 y en la Tabla 3.3. Además, en la misma tabla, se pueden observar unas aproximaciones del número de transferencias por segundo al número de bits por segundo. Aparte del aumento de velocidad existen cambios tanto en la codificación usada, como en el control de errores, el cual ha sido mejorado también en cada una de las versiones.

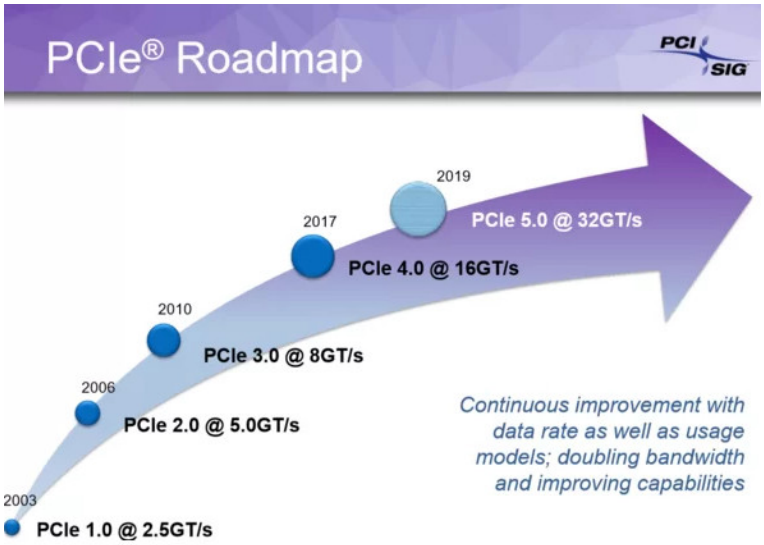


Figura 3.14 Evolución de PCI-e.

Tabla 3.3 Velocidades de las versiones de PCI-e.

Versión de PCI Express	Código en línea	Velocidad de transferencia	Ancho de banda (por carril)
1.0	8b/10b	2,5 GT/s	2 Gb/s (250 MB/s)
2.0	8b/10b	5 GT/s	4 Gb/s (500 MB/s)
3.0	128b/130b	8 GT/s	7.9 Gb/s (984,6 MB/s)
4.0	128b/130b	16 GT/s	15.8 Gb/s (1969,2 MB/s)

3.5 PXI: PCI eXtensions for Instrumentation

Se trata de una plataforma robusta para PC que ofrece una solución para sistemas de medición y automatización. PXI proporciona ventajas como el bajo coste, alto rendimiento y flexibilidad y los beneficios de un estándar abierto de la industria. PXI combina la tecnología de un PC estándar con el factor de forma de una especificación CompactPCI[2]. Existe un consorcio industrial que define los requisitos de hardware, electricidad, software, energía y refrigeración, lo que permite la interconectividad entre los dispositivos de diferentes fabricantes.

[2] Se trata de un conjunto de especificaciones que se definen a la hora de construir sistemas de forma modular, a bajo coste y de manera escalable. Siguen el estándar 3U y 6U de Eurocard en cuanto a las medidas.

3.5.1 PXI Hardware

- **Chasis PXI**
  El chasis PXI contiene un plano posterior de alto rendimiento que permite que las tarjetas en el sistema puedan comunicarse rápidamente entre sí. Como base de su sistema PXI que es, el chasis ofrece los buses de potencia, enfriamiento y comunicación de PCI y PCI Express para su controlador y sus módulos

Figura 3.15 Chasis PXI.

- **Controlador del sistema**
  El chasis PXI puede usar un controlador embebido de alto rendimiento con un SO Microsoft Windows o un SO en tiempo real (NI LabVIEW Real-Time) en el *Slot* 1 o un módulo que haga de interfaz para permitir la conexión a un controlador externo (como un PC). Esta última opción proporciona una opción muy rentable y poderosa. La elección del tipo de controlador PXI variará según la aplicación.

Figura 3.16 Controlador embebido.

- **Módulos**
  Existe una amplia gama en cuanto a módulos se refiere, incluyendo instrumentos de prueba que toman una amplia variedad de mediciones tales como voltaje, corriente, frecuencia, así como generadores de señales y formas de onda. También pueden realizar otras funciones, incluyendo adquisición de imágenes, fuentes de alimentación, conmutación y más.

Figura 3.17 Módulos PXI.

3.5.2 PXI Software

El estándar PXI depende de un entorno de software y hardware estandarizado. Como PXI se basa en el estándar PCI, muchas de las rutinas PCI se pueden aplicar al entorno PXI. Los módulos PXI no se pueden controlar desde un panel frontal físico, por lo tanto, se requiere el control del software a través del plano posterior. Los requisitos mínimos son Windows de 32 bits. Algunos proveedores también son compatibles con Linux u otros sistemas operativos. Los controladores IVI son opcionales, éstos son controladores de instrumentos sofisticados que presentan un mayor rendimiento y flexibilidad para aplicaciones de prueba más complejas que requieren intercambiabilidad, estado de almacenamiento en caché o simulación de instrumentos.

3.6 FireWire

Es una tecnología de entrada/salida de datos en serie a muy alta velocidad y para el conexionado de dispositivos digitales tales como videocámaras o cámaras digitales a ordenadores.

También conocido por el estándar que sigue, el IEEE 1394, es uno de los estándares más rapidos que existen por lo que es óptimo para transferencia de datos multimedia y otros dispositivos de alta velocidad como impresoras o discos duros.

Con FireWire se pueden alcanzar velocidades de hasta 400 Mb/s manteniéndola de forma estable. Además posee una flexibilidad de conexión bastante buena y es capaz de soportar hasta 63 dispositivos simultáneamente, aceptando longitudes de cable de máximo 4.25 m.

3.6.1 Versiones del estándar

- **FireWire 400 (IEEE 1394)**
  Puede transferir datos entre dispositivos a velocidades de datos semidúplex de 100, 200 o 400 Mb/s (las tasas de transferencia reales son 98.304, 196.608 y 393.216 Mb/s), conocidos como modos S100, S200 y S400 respectivamente. La longitud máxima de cable es 4.5 m aunque se pueden conectar en cadena hasta 16 cables con repetidores obteniendo un resultado de 72 m en total.

- **FireWire 800 (IEEE 1394b)**
  Mejora la velocidad de transferencia hasta los 800 Mb/s (786.5 Mb/s reales) con tecnología *full*-dúplex usando un nuevo esquema de codificación llamado beta, el cual se basa en 8b/10b. Además se extiende la distancia máxima de cable hasta los 100 m.

- **FireWire s800T (IEEE 1394c)**
  Proporciona mejoras técnicas las cuales permiten el uso de la interfaz con conectores RJ-45 con cables CAT 5. De esta manera se podían aprovechar las ventajas de Ethernet y FireWire simultáneamente, a pesar de ello, dejó de usarse.

- **FireWire s1600 y s3200**
  Permiten velocidades de 1.6 y 3.2 Gb/s (1.572864 Gb/s y 3.145728 Gb/s reales respectivamente) usando los mismos conectores de 9 pines que el FireWire 800.

3.6.2 Conectores

- **Cuatro contactos**
  Es considerablemente más pequeño que el de seis, pero la ausencia de dos contactos elimina la posibilidad de alimentar el componente conectado.

- **Seis contactos**
  Incluye salida de energía por lo que puede alimentar el dispositivo conectado sin la necesidad de usar una fuente de alimentación separada.

- **Nueve contactos**
  Es retrocompatible con las tasas de menor velocidad y con conectores FireWire 400. A pesar de ello, requiere tomas de corriente diferentes. En la Figura 3.18 se pueden observar los conectores FireWire y en la Tabla 3.4 el patillaje.

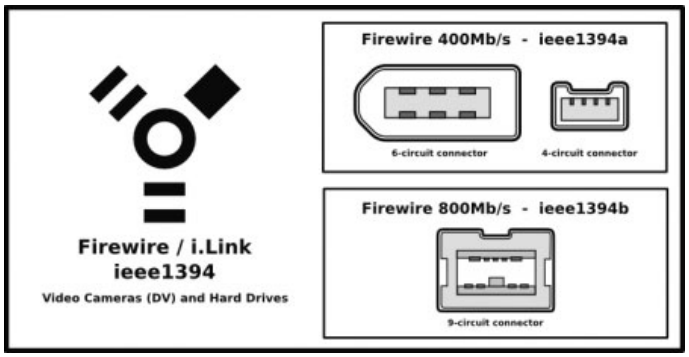


Figura 3.18 Conectores FireWire.

Tabla 3.4 Patillaje de conectores FireWire según su número de contactos.

Pin	Señal
1	TPB- (4 y 9 pines); Alimentación (6 pines)
2	TPB+ (4 y 9 pines); Tierra (6 pines)
3	TPA- (4 y 9 pines); TPB- (6 pines)
4	TPA+ (4 y 9 pines); TPB+ (6 pines)
5	TPA- (6 pines); A-shield (9 pines)
6	TPA+ (6 pines); Tierra (9 pines)
7	Sin conexión
8	Alimentación (9 pines)
9	B-shield (9 pines)

4 GPIB: General Purpose Interface Bus

El único error real es aquel del que no aprendemos nada.

Henry Ford

IEEE 488 es un estándar que define un bus de datos digital multimaestro de 8 bits paralelos para comunicaciones de corto alcance, desarrollado inicialmente por Hewlett-Packard en los años 70 como HP-IB (*Hewlett-Packard Interface Bus*).

A pesar de que se creó a finales de la década de los 60 para conectar equipos de prueba automatizados, también tuvo éxito en la década de los 70 y 80 como un bus periférico para las primeras microcomputadoras, como la Commodore PET. Los estándares más recientes han reemplazado al inicial IEEE 488 para su uso en computadoras, aunque se sigue usando en equipos de prueba y medida.

4.1 Estándares

En 1975, el IEEE estandarizó el bus como *Standard Digital Interface for Programmable Instrumentation*, IEEE 488, revisándose en 1978 y dando lugar a IEEE 488-1978. Posteriormente se revisó de nuevo en 1987, surgiendo el IEEE 488.1. En estos estándares se definió la parte mecánica, eléctrica y los parámetros básicos del protocolo de GPIB, sin especificar el formato de los datos y/o comandos.

En 1987, el IEEE introdujo *Standard Codes, Formats, Protocols, and Common Commands*, mejor conocido como IEEE 488.2. En este estándar se proporcionó la sintaxis básica y el convenio de formato, los comandos independientes de los dispositivos, estructuras de datos, protocolos de error y similares. IEEE 488.2 se basa en IEEE 488.1 pero no lo sustituye.

Como se ha explicado anteriormente, en IEEE 488.1 está definido el hardware y en IEEE 488.2 está definido el protocolo, sin embargo, los comandos no estaban estandarizados, por lo que diferían en instrumentos de la misma clase pero de distinto fabricante. Es por ello que HP desarrolló en 1989 su lenguaje TML, precursor de los comandos SCPI (*Standard Commands for Programmable Instruments*), los cuales fueron introducidos en 1990. SCPI agregó comandos genéricos estándar y una serie de comandos específicos para cada clase de instrumento.

El IEC (*International Electrotechnical Commission*) desarrolló sus propios estándares en paralelo a los del IEEE, con IEC 60625-1 e IEC 60625-2 (IEC 625), más tarde reemplazado por IEC 60488. Por otro lado, National Instruments introdujo en el mercado una extensión compatible con IEEE 488.1, originalmente conocida como HS-488. En ella se aumentó la velocidad máxima a 8 MB/s, disminuyendo ésta conforme se conectan más dispositivos al bus, dando lugar al IEEE 488.1-2003.
Finalmente, en 2004 el IEEE y el IEC combinaron sus respectivos estándares en: IEC 60488-1, *Standard for Higher Performance Protocol for the Standard Digital Interface for Programmable Instrumentation - Part 1: General*, el cual sustituye a IEEE 488.1 y a IEC 60625-1, y por otro lado, el IEC 60488-2,*Part 2: Codes, Formats, Protocols and Common Commands*, el cual sustituye a IEEE 488.2 y a IEC 60625-2.

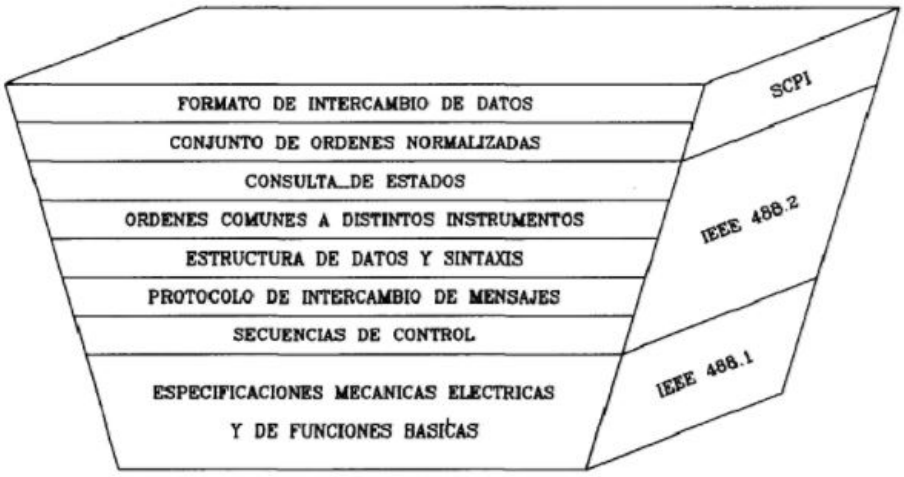


Figura 4.1 Relación entre las normas IEEE 488.1, IEEE 488.2 y SCPI.

4.2 Descripción general

El aspecto funcional de la norma incluye el uso de líneas específicas para enviar y recibir mensajes, el protocolo para la transmisión y recepción de esos mensajes, la lógica que utilizan y la temporización entre líneas. Además, se dispone de una serie de diez funciones de interconexión de propósitos específicos.

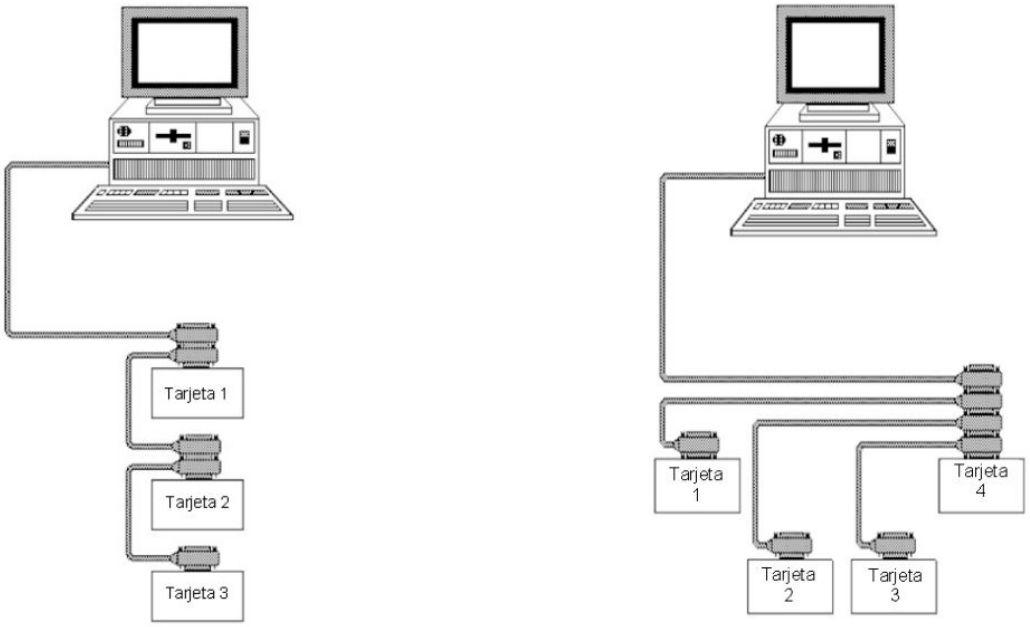
El bus transfiere órdenes y datos entre los dispositivos conectados al mismo a través de 16 líneas, las cuales se dividen en tres grupos:

- Bus de datos: Compuesto por 8 líneas (DIO1-DIO8) sobre las cuales los mensajes son transmitidos en forma de bytes consecutivos.
- Control de transferencia de bytes: Es un conjunto de tres líneas (DAV, NRFD y NDAC) cuya función es asegurar la correcta transmisión de los datos entre los equipos.
- Bus de control general de la interfaz: Constituido por 5 líneas (ATN, IFC, REN, SRQ y EOI) que se utilizan para enviar comandos entre los equipos relativos al modo de interpretar los datos transferidos o comandos básicos de manejo de la interfaz.

4.2.1 Limitaciones principales de GPIB

Las características mecánicas, eléctricas y funcionales de la norma imponen ciertas restricciones las cuales hay que cumplir para conseguir el funcionamiento óptimo del bus.

- Permite la interconexión de hasta 15 equipos, siendo uno de ellos el controlador del resto.
- Al menos dos tercios de los equipos conectados al bus deben estar encendidos.
- Los dispositivos conectados al bus pueden enviar y/o recibir información de cualquiera del resto de equipos.
- La tasa de transferencia máxima que se puede alcanzar es de 8 MB/s.
- La separación máxima entre dispositivos no debe ser mayor a 4 m, y la separación promedio en total de la red debe ser inferior a 2 m.
- La red no debe exceder la longitud máxima de 20 m.
- Se admite una configuración de la red en bus o en estrella, tal y como se muestra en la Figura 4.2.



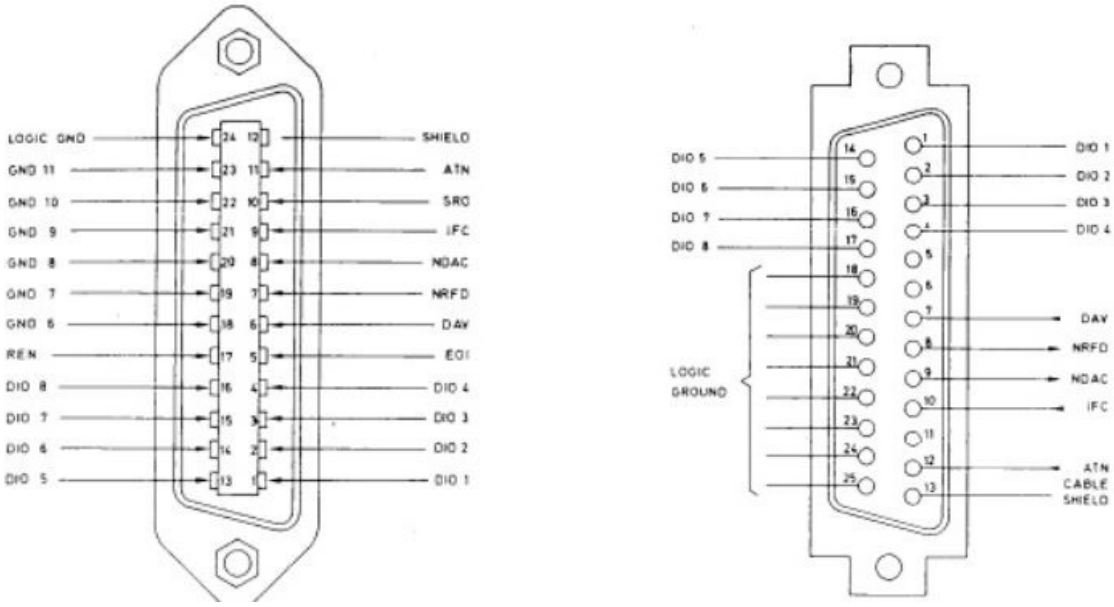
(a) Topología en bus. **(b)** Topología en estrella.

Figura 4.2 Formas de interconexión del bus GPIB.

A pesar de que solo se pueden conectar 15 dispositivos directamente, cada uno de ellos puede tener elementos secundarios, los cuales poseerían una dirección secundaria que se obtiene combinando la dirección principal con otro grupo de 31 direcciones para formar un total de 961 direcciones secundarias.

Por otro lado, es muy difícil obtener la velocidad máxima de transferencia, a no ser que se lleve a cabo un diseño muy cuidadoso de la red que tenga en cuenta los retardos digitales y los tiempos de establecimiento con cables de longitud mínima, cargas adecuadas y transceptores especiales.

Los conectores empleados pueden ser de tipo americano (24 terminales) o europeo (25 terminales), hembras en los instrumentos y macho y hembra en los cables (apilables), como se puede ver en la Figura 4.3.



(a) Conector GPIB americano. **(b)** Conector GPIB europeo.

Figura 4.3 Tipos de conectores GPIB.

4.3 Características eléctricas

Las especificaciones eléctricas dadas por la norma se basan en la tecnología TTL (*transistor-transistor logic*) y utilizan una lógica negativa, es decir, un nivel de tensión superior a 2.5 V se corresponde con un valor lógico *FALSE* y un nivel de tensión inferior a 0.8 V corresponde con un valor lógico *TRUE*.

Los integrados que manejan las señales SRQ, NRFD y NDAC deben ser *open-collector*, esto hará que una línea no adopte un valor *FALSE* hasta que todos los instrumentos fuercen dicho valor. Para las líneas DAV, IFC, ATN, REN y EOI pueden emplearse circuitos *open-collector* o *tristate* [1], útiles para alcanzar altas velocidades. Las líneas de datos, deben ser manejadas por circuitos *open-collector* solo en caso de que el dispositivo acepte la opción de encuesta paralelo.

En cuanto al dispositivo receptor, se prefiere el uso de dispositivos tipo Schmitt para todas las líneas de señal, lo que mejora la inmunidad al ruido. Cada línea de señal debe estar terminada internamente mediante una carga resistiva para mantener uniforme la impedancia de la línea.

4.4 Características funcionales

En este apartado se definen los tipos de instrumentos conectados al bus, sus órdenes, datos, direcciones y funciones básicas.

Se pueden considerar incluidos en ellas los protocolos de transferencia de información y el proceso del reconocimiento por parte del controlador de los instrumentos.

Los dispositivos están conectados en paralelo a las líneas de señal como se puede observar en la Figura 4.4, por lo que es necesario controlar la transmisión de los equipos para que no transmitan información varios simultáneamente. Para ello se emplean las líneas de *handshake* y se definen tres tipos de dispositivos:

- Transmisor (*TALKER*): capaz de transmitir datos a través del bus (p.e.: voltímetro).
- Receptor (*LISTENER)*: capaz de recibir datos digitales a través del bus (p.e.: fuente programable).
- Controlador (*CONTROLLER)*: capaz de administrar las comunicaciones a través del bus, designando los que han de transmitir y recibir en cada momento. Puede ordenar llevar a cabo instrucciones internas específicas a los dispositivos conectados. Un controlador puede ser de dos tipos:
  - *System controller*: capaz de tomar el control del bus en cualquier momento mediante las líneas IFC y REN. Solo puede existir uno en el bus.
  - *Active controller*: tiene la capacidad de enviar comandos para establecer los roles de receptores y transmisores, inicializar y sincronizar el bus y supervisar el estado de los equipos. Solo puede haber uno activo en cada momento a pesar de que pueden existir varios conectados capaz de asumir este papel.

  Según el número de controladores, los sistemas pueden ser de tres tipos:

  - Sin controlador: en el que solo un equipo es el emisor y el resto receptores, por lo que no es necesario controlador.
  - Con controlador único: donde el controlador puede mandar comandos y datos a los equipos, los equipos datos al controlador y también los equipos datos entre sí.
  - Con múltiples controladores: tiene las mismas capacidades que el caso anterior, sin embargo, incluye la característica de que es posible la transferencia de la capacidad de operar como *active controller*.
- *Idler*: sin ninguna capacidad respecto al bus.

[1] Tres estados: 0, 1 o alta impedancia.

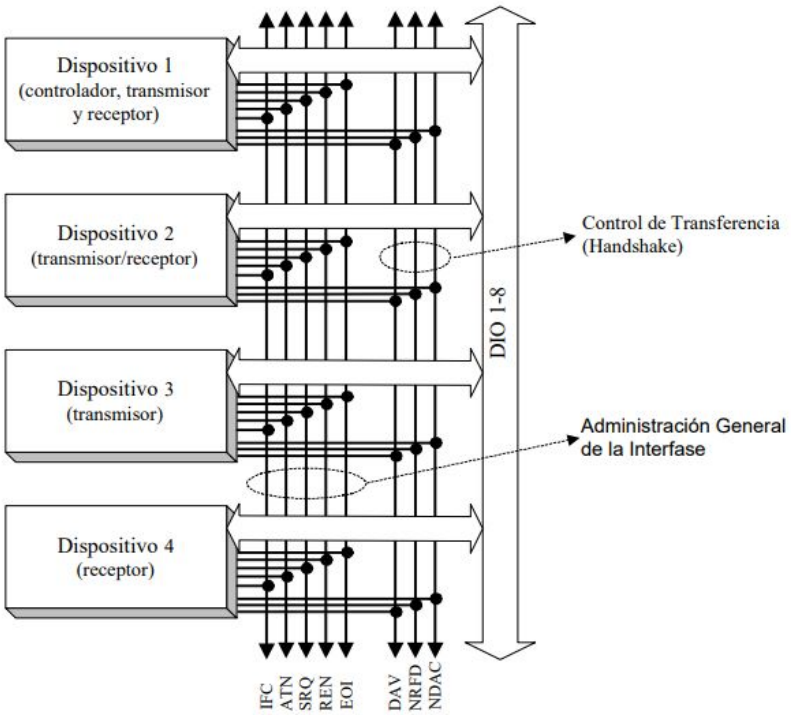


Figura 4.4 Diagrama de interconexión de dispositivos con el bus.

Para realizar esta conexión correctamente, hay que asignar una dirección a cada dispositivo conectado al bus, para que pueda ser identificado por el resto. Esta dirección es exclusiva dentro del mismo y consta de dos partes: dirección primaria (DP) y dirección secundaria (DS). La primera se trata de un número entero entre 0 y 30, indicando la dirección asignada; la segunda configura si el dispositivo se encuentra actuando como *talker* o como *listener*.

Posición del bit	7	6	5	4	3	2	1	0
Significado	0	TA	LA	Dirección primaria GPIB (de 0 a 30)				

Figura 4.5 Formato de las direcciones empleadas en el bus GPIB.

A la hora de la transmisión, el controlador habilita previamente quien será el receptor y el transmisor de dichos datos. Tras terminar la comunicación, el controlador podrá reasignar estos roles y podrá realizarse una nueva transmisión. En algunas ocasiones un controlador no es necesario, puesto que el transmisor y el receptor siempre será los mismos, por lo que la ausencia del controlador no supondrá ningún problema. Por otro lado, aunque pueden existir múltiples controladores en un sistema simultáneamente, solamente uno de ellos es el *controller-in-charge* (CIC), es decir, controlador encargado o principal.

4.4.1 Protocolo de transferencia del bus GPIB

El bus está compuesto por 16 líneas como se ha explicado anteriormente, agrupadas en tres grupos: 8 de bus de datos y 8 para comandos. De estas últimas, 3 son para el control de transferencia de datos (*handshake*) y las 5 restantes para el control general del bus. Esta distribución puede observarse en la Figura 4.4.

Líneas de datos

En las líneas DIO1-DIO8 se pueden transportar tanto datos como comandos en secuencia octetos (bytes). El bit DIO1 es el de menor peso (LSB) y el DIO8 el de mayor peso (MSB). Para que todos los dispositivos conectados sepan en todo momento de qué tipo de información se trata, el controlador pondrá en la línea ATN un valor verdadero si es un comando, en caso contrario, si son datos pondrá un valor falso.

Líneas de control de transferencia (*Handshake*)

Estas líneas ayudan a que todos los instrumentos conectados al bus reciban toda la información correctamente y no pierdan ningún dato. De esta manera, no se transmitirán datos hasta que el receptor más lento esté listo para ello.

- **DAV: *DAta Valid***
  Esta señal está controlada por el transmisor de los datos, la cual activa (nivel bajo) si dicha información que está en las líneas DIO es correcta y, por lo tanto, puede ser aceptada por los receptores.
- **NRFD: *Not Ready For Data***
  Esta señal está controlada por cada uno de los receptores. En el caso de que no se encuentre en condiciones de recibir datos, el receptor activará esta señal. Por otro lado, el transmisor deberá comprobar previamente que esta línea esté a nivel alto para activar la señal DAV. La línea NRFD está controlada por dispositivos a colector abierto, para realizar una lógica cableada y que solamente adopte el valor falso cuando todos los instrumentos estén listos.
- **NDAC: *Not Data* ACcepted**
  Esta señal está controlada por cada uno de los receptores. Se activa cuando todavía no se ha leído el octeto situado en las líneas de datos, por lo tanto, cuando todos hayan leído los datos, desactivarán esta señal y el transmisor podrá desactivar la señal DAV y cambiar los datos del bus.

La transferencia de un emisor se realiza siguiendo un procedimiento específico que se explicará a continuación. Tras la asignación por parte del controlador de las direcciones del emisor y de uno o más instrumentos receptores, coloca ATN a nivel alto (DIO contiene datos). Por su parte, los receptores ponen a nivel alto la señal NRFD cuando están preparados para recibir información. Una vez que el emisor detecta el nivel alto de dicha señal, coloca la información en el bus de datos y pone a nivel bajo la señal DAV, es decir, los datos están listos para leerse. Tras esto, una vez que los receptores lo han detectado, activan NRFD para impedir la emisión de un nuevo dato hasta que acepten el actual. Tras leer la información todos ellos ponen a nivel alto la señal NDAC, por lo que el emisor cambia la señal DAV a nivel alto para indicar que los datos no son válidos y forzar a los receptores a poner NDAC a nivel bajo. La secuencia se repite una vez estén listos de nuevo todos los receptores para recibir información.

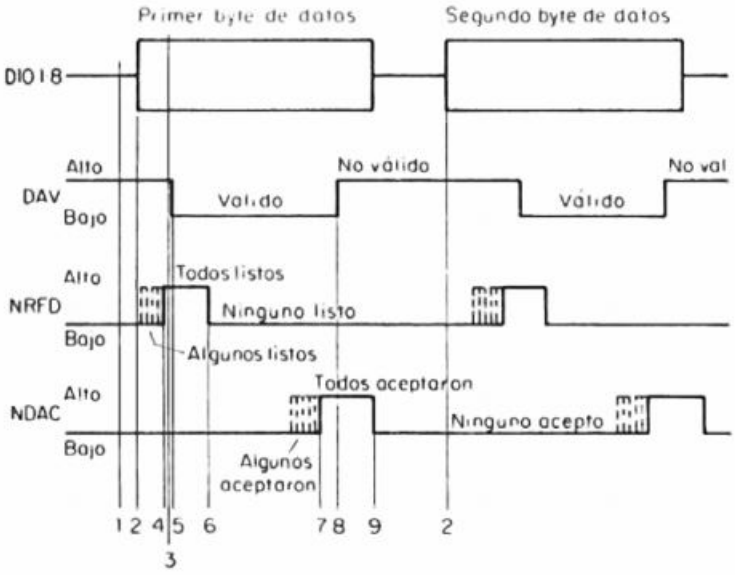


Figura 4.6 Diagrama de tiempos para las señales de *handshake*.

Líneas de control general

Las líneas pertenecientes a este grupo, controlan el flujo de información a través de la interfaz. También son conocidas como comandos multilínea.

- **ATN: *ATentioN***
  Es manejada por el controlador para indicar si el byte de las líneas DIO se trata de un comando (verdadero) o un dato (falso). El valor de esta señal se utiliza conjuntamente con la señal EOI.
- **EOI: *End Or Identify***
  Esta señal puede ser activada por un transmisor o un controlador. El primero la activa al mismo tiempo que coloca un byte en el bus de datos para indicar que es el último que se transfiere. El segundo la activa, simultáneamente a la señal ATN previamente explicada, para iniciar una identificación o sondeo en paralelo (*parallel poll*).
- **IFC: *InterFace Clear***
  Esta señal solamente la puede activar el controlador para llevar a todos los instrumentos a un estado conocido (inicialización). De esta manera se interrumpe el proceso que se esté realizando y desdirecciona los dispositivos.
- **SRQ: *Service ReQuest***
  Es utilizada por cualquier dispositivo conectado a la interfaz. Mediante esta señal, se indica la necesidad de atención, por lo que el controlador inicia un proceso de identificación o sondeo en secuencia (*serial poll*) para conocer cual es el instrumento que lo ha solicitado.
- **REN: *Remote Enable***
  Esta señal es activada por el controlador para indicar a los instrumentos conectados al bus que deben pasar al modo de control remoto. De esta forma, los dispositivos dejan de atender a los elementos de control situados en su panel frontal y solo admiten órdenes procedentes del bus.

Señales de masa

El resto de pines del conector son dedicados a señales de masa. Algunas de ellas tienen retornos de corrientes común y otras tienen un retorno propio, lo que provoca un aumento del número de líneas totales (8 masas).

- Retorno: Son las 6 líneas de masa (patillas 18, 19, 20, 21, 22 y 23 del conector americano) respecto a las cuales se miden respectivamente las señales DAV, NRFD, NDAC, IFC, SRQ y ATN.
- Masa del chasis (*SHIELD*): Es la patilla que se conecta a la masa del chasis del instrumento.
- Masa de referencia (*LOGIC GND*): Es la patilla de masa respecto a la cual se miden las señales EOI y REN.

4.4.2 Direccionamiento e identificación

El controlador selecciona a los instrumentos mediante el envío de su dirección por las líneas de datos (DIO). Cada instrumento la comparará con su propia dirección, produciéndose la identificación solo en uno de ellos. La dirección propia de un instrumento se elige mediante unos conmutadores accesibles desde el exterior o mediante un circuito programable colocado en su circuito interno.

Por otro lado, si un instrumento realiza una solicitud de servicio, es decir, activa la señal SRQ, el controlador deberá identificar al instrumento que la ha realizado, lo cual puede hacerlo de dos maneras diferentes:

- **Sondeo serie o en secuencia (*serial poll*)**
  El controlador direcciona uno tras otro los instrumentos conectados al bus, éstos responderan cada uno con un byte cuyo séptimo bit indica si ha solicitado el servicio o no. En caso afirmativo, los siete bits restantes son utilizados para definir el tipo de servicio (Figura 4.7). La consulta termina cuando el instrumento desactiva la señal SRQ.
- **Sondeo paralelo (*parallel poll*)**
  Se lleva a cabo cuando el controlador activa las señales ATN y EOI simultáneamente. En este modo, 8 instrumentos distintos envian 1 bit cada uno a las líneas de datos indicando si ha solicitado el servicio o no (Figura 4.8). Este método es más rápido, sin embargo, solo puede hacerse con 8 instrumentos a la vez.

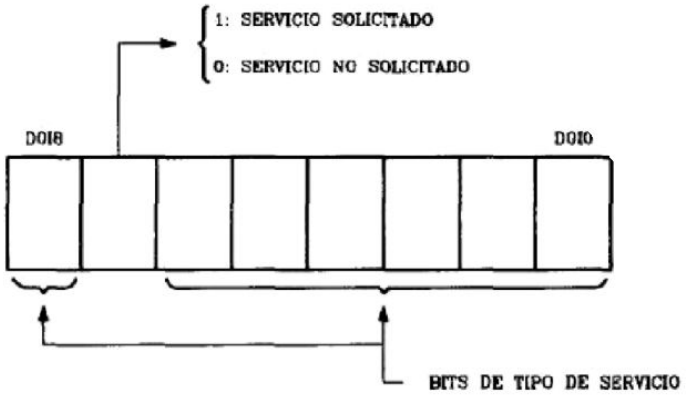


Figura 4.7 Byte de respuesta al controlador con sondeo serie.

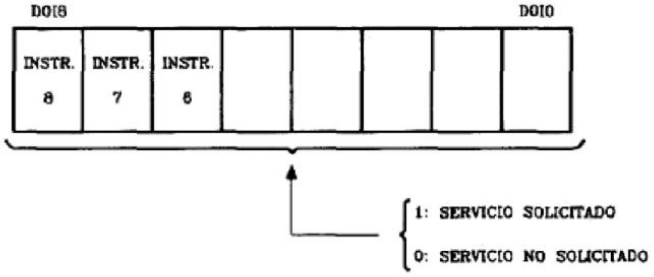


Figura 4.8 Byte de respuesta al controlador con sondeo paralelo.

4.4.3 Funciones básicas

Puesto que en la norma está definida la conexión de instrumentos que actúen como controladores, emisores o receptores, también se incluyen 10 funciones que utiliza el diseñador del instrumento para lograr las características deseadas del mismo.

Tabla 4.1 Funciones básicas de la norma IEEE 488.

Código simbólico	Denominación	Instrumentos
SH	Control de emisión (Source handshake)	Emisores y receptores
AH	Control de recepción (Acceptor handshake)	
T/TE	Emisión	
L/LE	Recepción	
SR	Petición de servicio (Service request)	
RL	Modo remoto o local (Remote/local)	Controlador
PP	Sondeo en paralelo (Parallel poll)	
DC	Inicialización (Device clear)	
DT	Disparo (Device trigger)	
C	Controlador (Controller)	

Cada función contiene un conjunto de mensajes particulares y está asignada a unos comandos concretos, los cuales se explicarán más adelante.
Las funciones se describirán a continuación brevemente:

1. Funciones básicas de los emisores y receptores:

   - Control de emisión (SH):
     Proporciona al instrumento la capacidad de poder transferir datos entre instrumentos mediante las líneas de control de transferencia.

   - Control de recepción (AH):
     Proporciona al instrumento la capacidad de poder aceptar datos de instrumentos empleando también las líneas de control de transferencia.

   - Emisión (T):
     Esta función concede la capacidad de enviar información a otros conectados al bus.

   - Recepción (L):
     Esta función concede la capacidad de recibir información procedente de otros conectados al bus.

   - Petición de servicio (SR):
     Proporciona la capacidad de solicitar atención al controlador activo.

2. Funciones básicas del controlador:

   - Modo remoto o local (RL):
     Permite al instrumento elegir entre dos fuentes de información: líneas de datos del bus (modo remoto) o el panel frontal (modo local). Es ejecutada activando la señal REN mediante el comando GTL.

   - Sondeo en paralelo (PP):
     Proporciona a un instrumento la capacidad de presentar un bit de estado al controlador en la línea previamente asignada. El controlador activa las líneas ATN y EOI y envía las órdenes UNT y PPC. Para finalizar el sondeo, envía las órdenes UNL, PPU y desactivando las señales previamente nombradas.

   - Inicialización (DC):
     Permite al controlador inicializar un instrumento activando ATN y enviando la orden SDC o DCL.

   - Disparo (DT):
     Provee a los instrumentos la capacidad para ejecutar una orden determinada, activando ATN y enviando la orden GET.

   - Controlador (C):
     Proporciona a un instrumento la capacidad de actuar como controlador (enviar direcciones, órdenes y ejecutar las peticiones de servicio de los instrumentos conectados o incluso realizar sondeos).

4.5 Comandos

Los comandos (mensajes *Command*) son enviados desde el controlador al resto de equipos para sincronizar o establecer su estado de operación.

En el envío de comandos, las líneas de datos son utilizadas si se requieren, con la diferencia de que en este caso la línea ATN es establecida a nivel bajo por el controlador para indicar que el dato volcado en el bus se trata de un comando. El comando es recibido por todos los equipos conectados al bus, independientemente de que sean *talker* o *listener*.

Los comandos se pueden clasificar en cinco grupos: *addressed*, *listen*, *talk*, *universal* y *secondary*. Los 7 bits menos significativos del bus son empleados para enviar los comandos, siendo los bits b7, b6 y b5 los que definen la naturaleza del comando enviado (Tabla 4.2).

Tabla 4.2 Naturaleza de los comandos.

b7	b6	b5	Tipo de comando
0	0	0	Addressed
0	0	1	Universal
0	1	X	Listen
1	0	X	Talk
1	0	X	Secondary

Cada equipo conectado al bus tiene una dirección exclusiva comprendida entre 0 y 30, establecida como ya se ha explicado anteriormente por hardware o por software. Esta dirección irá en los bits b4, b3, b2, b1 y b0 para referenciar al equipo. El código 0b11111, que corresponde con el 31, se emplea como dirección de difusión para referenciar a todos los equipos.

Comandos Talk/Listen

Con estos comandos, el controlador gestiona qué equipo actúa como *talker* o como *listener*. También se incluyen comandos que pasen a los equipos a modo *idler*.

Tabla 4.3 Comandos Talk/Listen.

Abreviatura	Nombre completo	Dato	Descripción
MTA	My Talk Address	0b010ddddd	Establece el modo *talker* en el equipo "ddddd"
UNT	Untalk	0b01011111	El equipo en el modo *talker* pasa a modo *idler*
MLA	My Listen Address	0b001ddddd	Establece el modo *listener* en el equipo "ddddd"
UNL	Unlisten	0b00111111	Todos los equipos en modo *listener* pasan a modo *idler*

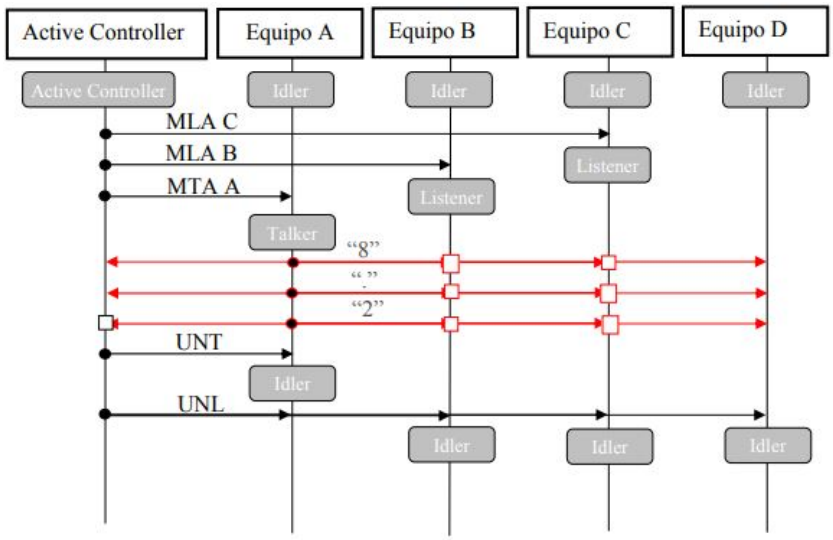


Figura 4.9 Ejemplo de uso de comandos Talk/Listen enviando un dato.

Para la finalización de los mensajes de datos existen tres métodos:

- Método EOI: tras el envío del último dato, el *talker* establece la línea del bus EOI al valor *true*.
- Método EOS: el último carácter enviado es uno establecido previamente.
- Método de cuenta: el controlador no permite enviar más bytes al *talker* cuando se ha alcanzado el número de bytes preestablecido en la cabecera del mensaje estableciendo las líneas NRFD y NDAC al estado lógico *true*.

Comandos Universales (UGC)

Son un conjunto de comandos que son enviados por el controlador a todos los equipos.

Tabla 4.4 Comandos Universales.

Abreviatura	Nombre completo	Dato	Descripción
LLO	Local Lockout	0b00010001	Se deshabilita el control mediante el panel frontal de los equipos
DCL	Device Clear	0b00010100	Inicializa las interfaces hardware y software de los equipos
PPU	Parallel Poll Unconfigure	0b00010101	Indica el final de una operación de sondeo en paralelo
SPE	Serial Poll Enable	0b00011000	Habilita a todos los equipos para que respondan al sondeo serie
SPD	Serial Poll Disable	0b00011001	Deshabilita el modo de sondeo serie

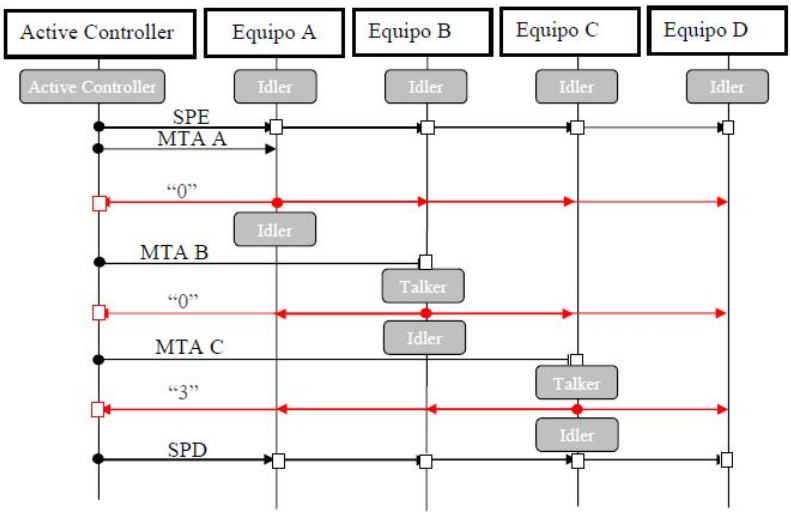


Figura 4.10 Ejemplo de uso de comandos *universal* realizando *serial poll*.

Comandos Addressed (ACG)

Los únicos destinatarios de estos comandos son los equipos que previamente han sido configurados como *listener*, por lo tanto solo les afecta a ellos.

Tabla 4.5 Comandos Addressed.

Abreviatura	Nombre completo	Dato	Descripción
GTL	Go To Local	0b00000001	Devuelve el control de los paneles frontales a los equipos
SDC	Selected Device Clear	0b00000100	Inicializa las interfaces hardware y software de los equipos *listener*
PPC	Parallel Poll Configure	0b00000101	Configura el inicio de una encuesta paralela quedando los equipos a la espera de un comando MSA
MSA	My Secondary Address	0b011xcbbb	Establece la línea (bbb) y el estado (c=0 significa request=FALSE) con la que responden a una encuesta paralela los equipos *listener*
GET	Group Trigger	0b00001000	Dispara el *trigger* de los equipos en estado *listener*. Permite operaciones sincronizadas entre instrumentos
TCT	Take Control	0b00001001	Establece como controlador activo al equipo que está direccionado como *listener*

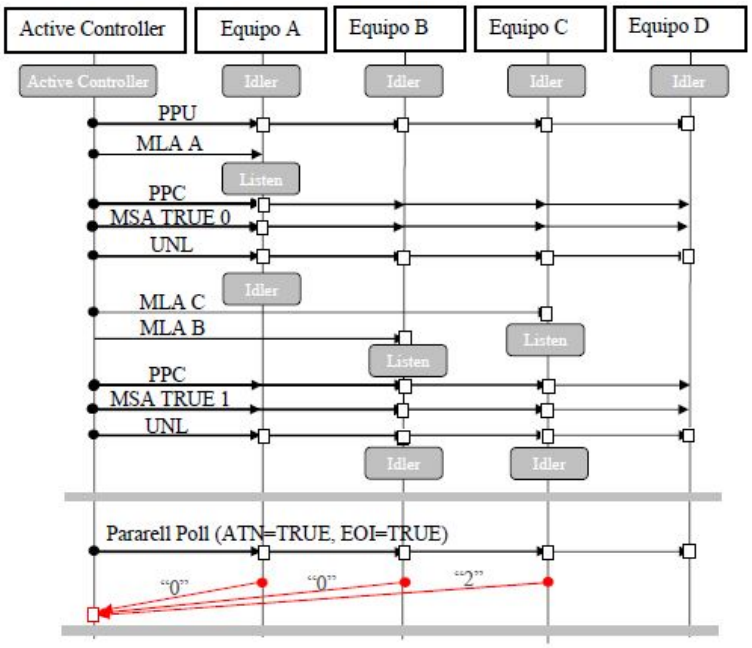


Figura 4.11 Ejemplo de uso de comandos *addressed* realizando *parallel poll*.

Comandos Comunes

Una innovación importante del estándar IEEE 488.2 fue la introducción de un conjunto estandarizado de comandos comunes para las muchas operaciones genéricas que todos los instrumentos deben realizar. El protocolo de intercambio de mensajes IEEE 488.2 es la base del estándar SCPI que hace que la programación del sistema de prueba sea aún más fácil. Los mnemónicos para estos comandos comunes comienzan con un asterisco (*) para diferenciarlos de los comandos específicos de cada dispositivo. Estos comandos se muestran en la Tabla 4.6.

Tabla 4.6 Comandos Comunes.

Mnemónico	Nombre completo	Función
*IDN?	Identification Query	Identifica el tipo de instrumento y versión del software
*RST	Reset Command	Sitúa al equipo en el estado básico de referencia
*TST?	Selft-Test Query	Requiere el resultado del *autotest* del equipo
*OPC	Operation Complete Command	Fija el bit de operación completa del registro estándar
*OPC?	Operation Complete Query	Responde con 1 si se han ejecutado órdenes previas
*WAI	Wait-to-Continue Command	Espera a que se realicen todas las operaciones pendientes
*CLS	Clear Status Command	Despeja el registro de estado y los registros de incidencia
*ESE	Event Status Enable Command	Habilita bits del registro de habilitación de incidencias
*ESE?	Event Status Enable Query	Interroga al registro de habilitación de incidencias estándar
*ESR?	Event Status Enable Query	Interroga al registro de incidencias estándar
*SRE	Service Request Enable Command	Habilita los bits del registro de habilitación de Byte de estado
*SRE?	Service Request Enable Query	Requiere el contenido del registro SER de habilitación del byte de estado
*STB?	Read Status Byte Query	Requiere el estado del registro resumido del Byte de estado
*LRN?	Learn Device Setup Query	Solicita el estado actual del equipo
*OPT?	Option Identification Query	Requiere la opción instalada en el equipo
*RCL	Recall Command	Restaura el estado del equipo del registro save/recall
*SAV	Save Command	Almacena el estado actual en un registro save/recall
*TRG	Trigger Command	Arranca o dispara la operación del equipo de forma remota

4.5.1 Secuencias de control y protocolos de un controlador

A pesar de que IEEE 488.2 tuvo menos impacto en los controladores que en los instrumentos, existen varios requisitos y mejoras opcionales para los controladores que hacen que un controlador IEEE 488.2 sea un componente necesario de los sistemas de prueba. IEEE 488.2 define con precisión la forma en que los

controladores envían comandos y datos y les agrega funcionalidad. Debido a estos requisitos del controlador IEEE 488.2, los fabricantes de instrumentos pueden diseñar instrumentos compatibles y eficientes. Los beneficios de esta estandarización para el desarrollador de sistemas de prueba son la reducción del tiempo y coste de desarrollo, ya que resuelve los problemas causados por las incompatibilidades de los instrumentos, las estructuras de comandos variables y los formatos de datos.

Secuencias de control IEEE 488.2

El estándar IEEE 488.2 definió secuencias de control que especifican los mensajes exactos que se envían desde el controlador, así como el orden de múltiples mensajes. IEEE 488.2 definió 15 secuencias de control obligatorias y cuatro secuencias de control opcionales, como se muestra en la Tabla 4.7.

Tabla 4.7 Secuencias de control obligatorias y opcionales de un controlador de la norma IEEE 488.2.

Secuencia de control	Carácter	Descripción
SEND COMMAND	Obligatorio	Envío de órdenes con ATN en nivel alto
SEND SETUP	Obligatorio	Envío de dirección para enviar datos
SEND DATA BYTES	Obligatorio	Envío de datos con ATN en nivel bajo
SEND	Obligatorio	Envío de un mensaje del programa
RECEIVE SETUP	Obligatorio	Envío de dirección para recibir datos
RECEIVE RESPONSE MESSAGE	Obligatorio	Recepción de datos con ATN en nivel bajo
RECEIVE	Obligatorio	Recepción de un mensaje
DEVICE IFC	Obligatorio	Envío de señal de inicialización de instrumentos IFC
DEVICE CLEAR	Obligatorio	Puesta en estado inicial de los dispositivos
ENABLE LOCAL CONTROLS	Obligatorio	Puesta de los dispositivos en estado de control local
ENABLE REMOTE	Obligatorio	Puesta de los dispositivos en estado de control remoto
SET RWLS	Obligatorio	Puesta de los dispositivos en estado de control remoto e inhibición del control local
SEND LLO	Obligatorio	Puesta de los dispositivos en estado de control local e inhibición del control remoto
READ STATUS BYTE	Obligatorio	Lectura del octeto de estado de la norma IEEE 488.1
TRIGGER	Obligatorio	Envío de mensaje de inicio de ejecución de un grupo de acciones
PASS CONTROL	Opcional	Paso del control a otro dispositivo
PERFORM PARALLEL POLL	Opcional	Realización de un sondeo en paralelo
PARALLEL POLL CONFIGURE	Opcional	Configuración o desinhibición de los dispositivos para realizar un sondeo en paralelo
PARALLEL POLL UNCONFIGURE	Opcional	Inhibición de los dispositivos para realizar un sondeo en paralelo

Protocolos IEEE 488.2

Los protocolos son rutinas de alto nivel que combinan varias secuencias de control para realizar operaciones comunes del sistema de prueba. IEEE 488.2 define dos protocolos obligatorios y seis protocolos opcionales, como se muestra en la Tabla 4.8.

Tabla 4.8 Protocolos obligatorios y opcionales de un controlador de la norma IEEE 488.2.

Palabra clave	Carácter	Función
RESET	Obligatorio	Inicialización del sistema
FINDRQS	Opcional	Identificación del dispositivo que solicita servicio
ALL SPOLL	Obligatorio	Sondeo secuencial de todos los dispositivos
PASS CTL	Opcional	Cesión del control
REQUEST CTL	Opcional	Petición de control
FINDLSTN	Opcional	Búsqueda de receptores
SETADD	Opcional (necesita FINDLSTN)	Establecimiento de direcciones
TESTSYS	Opcional	Autoverificación del sistema

4.5.2 Comandos SCPI

SCPI (*Standard Commands for Programmable Instruments*) es un estándar que define un conjunto de mensajes que forman un lenguaje que permite la comunicación con instrumentos. SCPI es muy usado por la mayoría de instrumentos GPIB, sin embargo, también es muy usado en RS-232, Ethernet, PXI o VXI. La adoptación de este lenguaje por varios de los grandes fabricantes como HP o Agilent ha permitido cierta uniformidad en las comunicaciones entre dispositivos.

SCPI va mas allá de 488.2 y define un estándar de comandos de programación. Por ejemplo, para una función de medida determinada, SCPI define los comandos específicos que hay que emplear para acceder a ella a través de las interfaces LAN, GPIB o USB. Con SCPI se logran dos ventajas muy importantes:

- Si se sabe utilizar unas funciones en un equipo SCPI, se sabrá controlar las mismas funciones en otro equipo SCPI distinto.
- Un programa escrito para un instrumento SCPI, se adaptará fácilmente a otro SCPI.

Los comandos SCPI no son más que cadenas ASCII que se envían al instrumento sobre la capa física, mediante esto, se pueden realizar operaciones o consultas (éstas últimas caracterizadas por el símbolo "**?**") El formato de los mensajes en su forma canónica es, aplicándolo a un ejemplo: **"TRIGger <m>: LEVel <n> [: VALue] <Level>"**, teniendo en cuenta que:

1. Las partes entre corchete son opcionales.
2. La parte de las mayúsculas es obligatoria, al contrario que la de las minúsculas que es opcional.
3. Los comandos SCPI no distinguen entre mayúsculas y minúsculas.
4. Se pueden combinar comandos usando el punto y coma. Por ejemplo: **TRIG1: SOUR CH1"** y **"TRIG1: LEV2 3.5"** es igual que **"TRIG1: SOUR CH1; LEV2 3.5"**. Esto se debe a que la ruta del árbol de comandos no cambia dentro de una cadena a no ser que, tras el punto y coma, se introduzcan dos puntos (**"TRIG1: SOUR CH1;: CHAN2: STATe ON"**).
5. Los parámetros numéricos sin unidades se entenderán en unidades del Sistema Internacional (SI).

4.5.3 La norma 488.2

El estándar 488.2 define los modos de operación básicos de los equipos tales como el intercambio de mensajes entre el controlador y el instrumento, como por ejemplo, cuando están dispuestos a escuchar y hablar a otros equipos y qué ocurre cuando no se cumple el protocolo. El modelo de un equipo que cumple la norma 488.2 se puede representar como se muestra en la Figura 4.12.

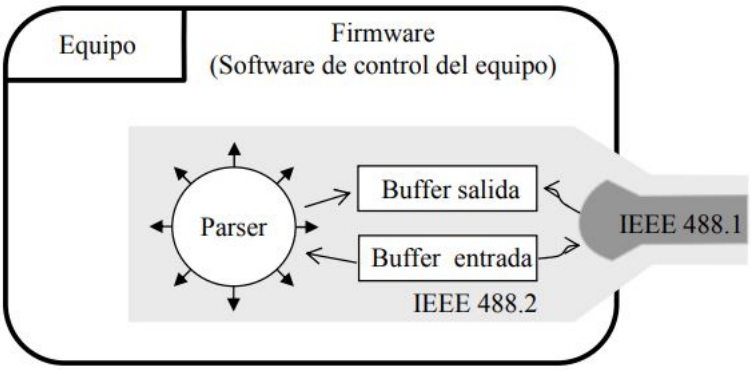


Figura 4.12 Modelo de intercambio de un equipo 488.2.

Búffer de salida
Zona de la memoria donde las órdenes son almacenadas antes de ejecutarse por parte del *parser*. De esta manera, se pueden almacenar varias órdenes contenidas en un *string* y que el controlador se libere y pueda comunicarse con otros equipos.

Búffer de entrada
Zona de la memoria en la que los datos se almacenan a la espera de que el controlador los lea.

Parser
Se trata del controlador interno que interpreta los mensajes recibidos por el *buffer* de entrada y los ejecuta. En caso de necesitar una respuesta por parte del instrumento, coloca los datos correspondientes en el *buffer* de salida.

Protocolo básico

- El equipo y el controlador se intercambian mensajes de órdenes y de respuestas; los primeros pueden ser de control, en caso de que requieran un cambio del estado del equipo o de requerimiento, en caso de solicitar información acerca del estado del equipo. Hasta que la respuesta a dicho requerimiento no llega al controlador, éste no envía más órdenes.
- La regla básica del protocolo es:
  "El equipo solo habla cuando está dispuesto a ello, y en ese caso, tiene que hablar antes de que se le ordene hacer una cosa nueva".
- Cuando el equipo es encendido o recibe una orden **CLS* ambos *buffers* son inicializados, así como el *parser* hace lo propio a la raíz de su árbol de órdenes.
- El equipo y el controlador se intercambian mensajes de órdenes y de respuestas completos, por lo que el controlador deberá enviar un mensaje completo de órdenes antes de intentar leer una respuesta. De igual forma, cuando le llegue un mensaje de respuesta, deberá leerlo completo antes de enviar una nueva orden.
- El controlador puede mandar un requerimiento compuesto, lo cual consiste en enviar un mensaje con múltiples órdenes de requerimientos en él separados por el carácter ";". Los mensajes de respuesta saldrán por el *buffer* en el mismo formato.

Protocolos de excepción

Si se produce un fallo en el intercambio de información entre equipos, el proceso no termina de forma normal sino que existe un protocolo de excepción el cual se lleva a cabo. Los distintos casos que pueden ocurrir son:

- Equipo direccionado para hablar sin nada en cola.
  - Si el fallo producido es que el instrumento no ha recibido la orden de requerimiento, éste indicará un error de encolamiento y no enviará ningún byte por el bus.
  - Sin embargo, si el fallo es producido por no haber podido ejecutar la orden de requerimiento (a consecuencia de otro error), el equipo no indica fallo de encolamiento y espera a recibir un mensaje del controlador.
- Equipo direccionado para hablar sin que nadie escuche.
  El equipo direccionado como *talker* esperará a que algún equipo sea direccionado como *listener* o bien a que el controlador tome el control del bus.
- Error de orden.
  Es producido cuando existe un fallo en la sintaxis ya que el equipo no podrá interpretar la orden correctamente al no reconocerla.
- Error de ejecución.
  Puede producirse en dos tipos de situaciones. Por un lado, si alguno de los parámetros está fuera del rango establecido y, por otro lado, si el equipo se encuentra en un estado en el que la ejecución del comando introducido no esté permitida.
- Error específico de equipo.
  Si el equipo en cuestión no puede ejecutar una orden por cualquier motivo dependiente única y exclusivamente de sí mismo y no como consecuencia del protocolo, se produce este error.
- Error de encolamiento.
  Se produce si no se sigue el protocolo de lectura de los datos de la cola de salida.
- Condición inconclusa.
  Se genera en caso de que el controlador intente leer un mensaje de respuesta antes de que el programa haya terminado de generar dicho mensaje.
- Condición interrumpida.
  En caso de que el controlador envíe un mensaje de orden antes de terminar de leer por completo una respuesta producida por un equipo ante un mensaje de requerimiento, el equipo genera un error de encolamiento y el segmento de mensaje no leído es eliminado del *buffer* de salida.

- Bloqueo de *buffer*
  Se produce al estar lleno el *buffer* de entrada, lo que provoca que el de salida también se llene. Si un controlador envía un mensaje con muchas órdenes de requerimiento, entonces el equipo generará un mensaje de respuesta muy largo, el cual puede superar la capacidad del *buffer* de salida. El controlador no puede terminar de enviar el mensaje al estar la entrada llena, la cual no se vacía hasta que la respuesta no termine de ejecutarse. Para solucionarlo, el equipo limpia la cola de salida y genera un error de encolamiento.

Reporte de estado

Otra innovación del estándar IEEE 488.2 es un esquema estandarizado para informes de estado. Este sistema de informe de estado está disponible para informar sobre eventos significativos que ocurren dentro de cada instrumento conectado al bus. En este esquema, cada instrumento está equipado con dos registros de estado, llamados *Event Status Register* (SESR) y *Status Byte Register* (SBR). Cada bit en estos registros corresponde con un tipo particular de evento que puede ocurrir mientras el instrumento está en uso, como un error de ejecución o la finalización de una operación. Cuando se produce el evento de un tipo dado, el instrumento establece el bit del registro de estado asociado en un valor de uno, si ese bit se ha habilitado previamente en el registro correspondiente. Por lo tanto, leyendo los registros de estado, es posible decir qué eventos han ocurrido.

El SESR, que se muestra esquemáticamente en la Tabla 4.9, registra ocho tipos de eventos que pueden ocurrir dentro de un instrumento de adquisición de datos.

Tabla 4.9 Bits del Standard Event Status Register (SESR).

Standard Event Status Register (SESR)							
7	6	5	4	3	2	1	0
PON	URQ	CME	EXE	DDE	QYE	RQC	OPC

Las funciones de estos bits se describen en la Tabla 4.10.

Tabla 4.10 Funciones de los bits del Standard Event Status Register (SESR).

Bit	Función asociada
7 (MSB)	Power On: indica el encendido del instrumento.
6	User Request: un botón del panel frontal se ha pulsado.
5	Command Error: el instrumento recibió un comando con sintaxis errónea.
4	Execution Error: un error ha ocurrió mientras el instrumento ejecutaba un comando.
3	Device Dependant Error: indica que el dispositivo está funcionando incorrectamente.
2	Query Error: se intentó leer el mensaje de salida cuando no había datos o se recibió un nuevo comando antes de que se leyeran los datos previamente solicitados.
1	Request Control: el instrumento solicita ser controlador.
0 (LSB)	Operation Complete: todos los comandos han sido completados.

El SESR se emplea como una herramienta de señalización de eventos. Al iniciar las comunicaciones con los instrumentos, es necesario activar dichos bits de notificación que interesen. Esto, para los instrumentos que cumplen el estándar IEEE 488.2, se realiza mediante el comando **ESE* (*Event Status Enable*). Por ejemplo, si se desea activar el bit QYE y así detectar cuando ocurre un error en el bit 2 del SESR, habrá que escribir **ESE 4* (4=0b00000100).

El SBR, que se muestra esquemáticamente en la Tabla 4.11, registra si los datos están disponibles en el *buffer* de salida del instrumento, si el instrumento solicita el servicio y si el SESR ha registrado algún evento. Las funciones de los bits del registro se describen en la Tabla 4.12.

Tabla 4.11 Bits del Status Byte Register (SBR).

Status Byte Register (SBR)							
7	6	5	4	3	2	1	0
-	RQS	ESB	MAV	-	-	-	-

Tabla 4.12 Funciones de los bits del Status Byte Register (SBR).

Bit	Función del bit
7 (MSB)	Puede ser definido para el uso por el fabricante del instrumento.
6	Request Service: el instrumento requiere de los servicios del controlador.
5	Event Status Byte: es activado si algún evento del registro SESR ocurre.
4	Message Available: hay un dato disponible en el *buffer* de salida.
3-0	Puede ser definido para el uso por el fabricante del instrumento.

Un instrumento se puede configurar para poder realizar peticiones de servicio (RQS), que es una señal digital por un cable dedicado del bus GPIB de 24 hilos en respuesta a cualquiera de los siguientes dos eventos: un evento detectado por el SESR o la presencia de datos solicitados en el *buffer* de salida (es decir, activación de bits ESB o MAV, respectivamente). Para configurar esto, se hace a través del comando **SRE* (*Service Request Enable*).

Por ejemplo, si se desea que se realice una peticion de servicio si un evento en el SESR ha ocurrido, habrá que activar el bit ESB del registro SBR por lo que habrá que escribir: **SRE 32* (32=0b00100000). Sin embargo, si se quiere activar la petición de servicio cuando haya datos en el *buffer* de salida, lo que se deberá escribir es: **SRE 16* (16=0b00010000). La relación entre el SESR, el *buffer* de salida y el SBR se muestra en la Figura 4.13.

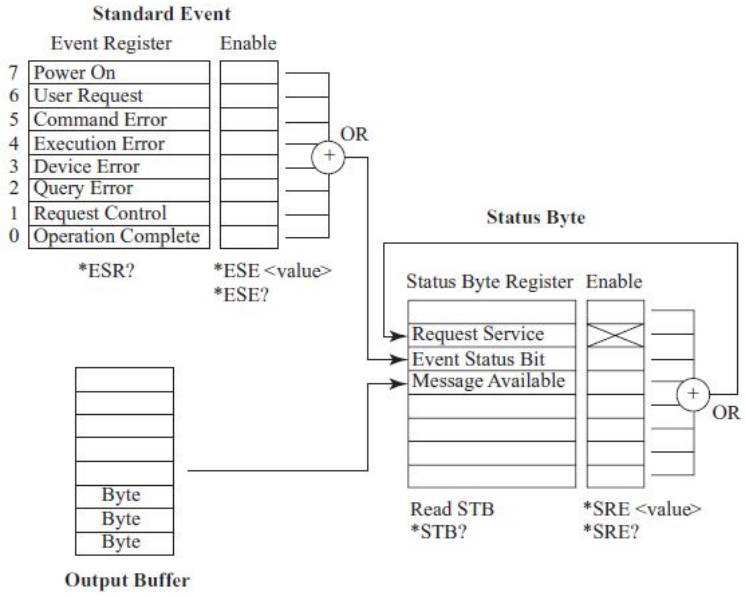


Figura 4.13 Registros de estado del estándar IEEE 488.2.

5 Equipo utilizado

No dejes que el ruido de las opiniones de otros apague tu propia voz interior.

STEVE JOBS

Para realizar este Trabajo Fin de Grado, el equipo de laboratorio que será controlado remotamente es la fuente de alimentación HP 6622A (Figura 5.1). Dicha fuente ofrece unas especificaciones y precisión que nos permiten una optimización de las medidas. Sus principales ventajas son:

- Dos salidas de la fuente completamente aisladas en un rack de 3U.
- Salidas de doble rango.
- Mediciones incorporadas y características avanzadas programables.
- Salidas rápidas y con bajo ruido.
- Funciones de protección para garantizar la seguridad del DUT.
- Compatible con GPIB.

Figura 5.1 Fuente de alimentación HP 6622A.

5.1 Especificaciones

5.1.1 Rangos de salida

Para ambos canales, a diferencia de otras fuentes que se pueden encontrar en el mercado, los rangos de salida son los mismos:

- Bajos voltajes, amperaje: 0-20 V, 0-4 A.
- Altos voltajes, amperaje: 0-50 V, 0-2 A.

El máximo valor programable de voltaje para cada rango es un 1 % mayor que el voltaje nominal y el máximo valor programable de corriente para cada rango es un 3 % mayor que la corriente nominal.

5.1.2 Precisión

Los siguientes valores están dados para una temperatura de 25°C± 5°C.

Voltaje: 0.06 % + 50 mV.

Corriente: 0.16 % + 40 mA.

5.1.3 Rizado y ruido

Voltaje eficaz: 500 μV.

Voltaje pico a pico: 3 mV.

Corriente eficaz: 1 mA.

5.1.4 Control remoto

Para controlar la fuente remotamente solo se necesitarán un PC operativo y la fuente de alimentación conectados al bus GPIB. La manera en la que se le mandarán órdenes a dicha fuente es, como se ha visto anteriormente, mediante comandos SCPI.

GPIB

El control remoto es implementado mediante GPIB, el cual permite mandarle instrucciones usando un ordenador externo equipado con una interfaz GPIB. La fuente de alimentación implementa las funciones IEEE 488 siguientes:

SH1: Source Handshake.

AH1: Acceptor Handshake.

T6: Talker.

L4: Listener.

SR1: Service Request.

RL1: Remote/Local.

RL1: Parallel Poll.

DC1: Device Clear.

Sintaxis

En la Figura 5.2 se puede ver el formato de los comandos SCPI con los que se va a trabajar en el presente proyecto.

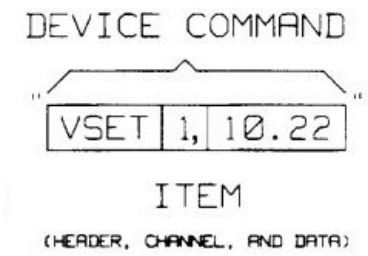


Figura 5.2 Formato de los comandos SCPI.

En la Tabla 5.1 se pueden observar los comandos empleados para programar la fuente de alimentación. Por otro lado, en la Tabla 5.2 se pueden observar las consultas a la misma.

Tabla 5.1 Comandos de la fuente 6622A.

Command	Header	Output Channel	Data Range (Fig. 5-2)	Syntax
Query accumulated status register	ASTS?	1, 2, 3, 4	---	Q2
Return power supply to turn on state	CLR	---	---	C1
Turn calibration mode on or off	CMODE	---	0,1 (OFF,ON)	C2
Query if calibration mode is on	CMODE?	1, 2, 3, 4	See Table 5-4	Q1
Set the state of outputs at power on	DCPON	---	0,1 (OFF, ON) CC+ 2,3 (OFF, ON) CC-	
Query outputs power on state	DCPON?	---	---	
Set the reprogramming delay time	DLY	1, 2, 3, 4	0-32	C4
Query the setting of the delay time	DLY?	1, 2, 3, 4	---	Q2
Turn front panel display on or off	DSP	---	0,1(OFF,ON)	C2
Display a string on front panel	DSP	---	"STRING"	C6
Query if front panel display is on	DSP?	---	---	Q1

Command	Header	Output Channel	Data Range (Fig. 5-2)	Syntax
Query present hardware error	ERR?	---	See Table 5-8	Q1
Query fault register	FAULT?	1, 2, 3, 4	---	Q2
Query the model number of supply	ID?	---	---	Q1
Program the I DAC in counts	IDAC	1, 2, 3, 4	See Service Manual	C4
Query setting of I DAC in counts	IDAC?	1, 2, 3, 4	---	Q2
Send data to calibrate I circuits	IDATA	1, 2, 3, 4	See Table 5-4	C5
Sets output to high I cal. value	IHI	1, 2, 3, 4	See Table 5-4	C3
Sets output to low I cal. value	ILO	1, 2, 3, 4	See Table 5-4	C3
Query measured I output	IOUT?	1, 2, 3, 4	See Table 5-4	Q2
Used to Calibrate I readback circuits	IRDAT	1, 2, 3, 4	See Table 5-4	C5
Set output to + I readback high cal value	IRHI	1, 2, 3, 4	See Table 5-4	C3
Set output to - I readback high cal value	IRHN	1, 2, 3, 4	See Table 5-4	C3
Set output to - I readback low cal value	IRLN	1, 2, 3, 4	See Table 5-4	C3
Set output to + I readback low cal value	IRLO	1, 2, 3, 4	See Table 5-4	C3
Set full scale current range of output	IRSET	1, 2, 3, 4	See Table 5-4	C4
Query full scale current range	ISET?	1, 2, 3, 4	See Table 5-4	Q2
Set current of an output	ISET	1, 2, 3, 4	See Table 5-4	C4
Query current of an output	ISET?	1, 2, 3, 4	See Table 5-4	Q2
Increase or decrease output current by value	ISTEP	1, 2, 3, 4	See Table 5-4	C4
Select which output will be metered	METER	---	1-4	C2
Query which output is being metered	METER?	---	---	Q1
Send data to calibrate -I readback	NIDAT	1, 2, 3, 4	See Table 5-4	C5
Enable overcurrent protection	OCP	1, 2, 3, 4	0,1 (OFF,ON)	C4
Query if OCP is enabled	OCP?	1, 2, 3, 4	---	Q2
Reset overcurrent protection	OCRST	1, 2, 3, 4	---	C3
Turn output on or off	OUT	1, 2, 3, 4	0,1 (OFF,ON)	C4
Query if output is on or off	OUT?	1, 2, 3, 4	---	Q2
Perform overvoltage calibration	OVCAL	1, 2, 3, 4	See Table 5-4	C3
Reset overvoltage circuit	OVRST	1, 2, 3, 4	---	C3
Set overvoltage trip value	OVSET	1, 2, 3, 4	See Table 5-4	C4
Query overvoltage trip value	OVSET?	1, 2, 3, 4	See Table 5-4	Q2
Enable power on service request	PON	---	0,1 (OFF,ON)	C2
Query if PON is enabled	PON?	1, 2, 3, 4	---	Q1
Recall voltage and current settings	RCL	---	0-10	C2
Set readback DAC to a number of counts	RDAC	1, 2, 3, 4	See Service Manual	C4
Query readback DAC count setting	RDAC?	1, 2, 3, 4	---	Q2
Query revision date of ROM	ROM?		---	Q1
Query revision date of secondary ROM	SROM?	1, 2, 3, 4	---	Q2
Set causes for generating a service request	SRQ	---	01,2,3	C2
Query causes which will generate an SRQ	SRQ?	---	---	Q1

Command	Header	Output Channel	Data Range (Fig. 5-2)	Syntax
Store present output state	STO		0-10	C2
Query preset status of output	STS?	1, 2, 3, 4		Q2
Perform self test on GP-IB interface	TEST?			C1
Set bits in mask register	UNMASK	1, 2, 3, 4	0-255	C4
Query bits set in mask register	UNMASK?	1, 2, 3, 4		Q2
Program the voltage DAC in counts	VDAC	1, 2, 3, 4	See Service Manual	C4
Querys setting of voltage DAC in counts	VDAC?	1, 2, 3, 4		Q2
Send data to calibrate the voltage circuits	VDATA	1, 2, 3, 4	See Table 5-4	C5
Set output to high V calibration value	VHI	1, 2, 3, 4	See Table 5-4	C3
Set output to low V calibration value	VLO	1, 2, 3, 4	See Table 5-4 V	C3
Query inputs of analog multiplexer	VMUX?	1, 2, 3, 4	1-18	C4
Query measured value of an output	VOUT?	1, 2, 3, 4	See Table 5-4	Q2
Calibrate the voltage readback circuitry	VRDAT	1, 2, 3, 4	See Table 5-4	C5
Set output to V readback high cal value	VRHI	1 2, 3, 4	See Table 5-4	C3
Set output to V readback low cal value	VRLO	1, 2, 3, 4	See Table 5-4	C3
Set full scale voltage programming range	VRSET	1, 2, 3, 4	See Table 5-4	C4
Query full scale voltage programming range	VRSET?	1, 2, 3, 4	See Table 5-4	Q2
Set output voltage	VSET	1, 2, 3, 4	See Table 5-4	Q2
Query setting of output voltage	VSET?	1, 2, 3, 4	See Table 5-4	Q2
Increase or decrease output voltage by value	VSTEP	1, 2, 3, 4	See Table 5-4	C4

Tabla 5.2 *Queries* de la fuente 6622A.

Query	Header (Note 7)	Channel (Note 1)	Response (Notes 5 and 6)	Initial Value	Syntax (Fig. 5-2)
Voltage Setting	VSET?	1,2,3,4	SZD.DDD(Note 3)	0(Note 8)	Q2
Current Setting	ISET?	1,2,3,4	(Note 2)	10 mA (Note 8)	Q2
Full Scale Current Range	IRSET?	1,2,3,4	(Note 2)	High (Note 8)	Q2
Full Scale Voltage Range	VRSET?	1,2,3,4	ZD.DDD (Note 8)	High (Note 8)	Q2
Voltage Output	VOUT?	1,2,3,4	SZD.DDD (Note 3)	--	Q2
Current Output	IOUT?	1,2,3,4	(Note 2)	--	Q2
OVP Setting	OVSET?	1,2,3,4	SZZD.DD	55 V (Note 8)	Q2
OC Protection On/Off	OCP?	1,2,3,4	ZZD	--	Q2
Output On/Off	OUT?	1,2,3,4	ZZD	--	Q2
Unmask Setting	UNMASK?	1,2,3,4	ZZD	0 (Note 8)	Q2
Delay Setting	DLY?	1,2,3,4	<sp>ZD.DDD	.02 (Note 8)	Q2
Status	STS?	1,2,3,4	ZZD	--	Q2
Accumulated Status	ASTS?	1,2,3,4	ZD	--	Q2
Fault	FAULT?	1,2,3,4	ZZD	--	Q2
Error	ERR?	--	ZZD	--	Ql
Service Request Setting	SRQ?	--	ZZD	0 (OFF)	Q1
Power-On SRQ On/Off	PON?	--	ZZD	0 (OFF) (Note 9)	Q1
Display On/Off	DSP?	--	ZZD	1 (ON)	Q1
Model Number	ID?	--	Agilent 662XA (Note 4)	--	Q1
Selftest	TEST?	--	ZZD	--	Q1
Calibration Mode	CMODE?	--	ZZD	0 (OFF)	Q1
DC Power On	DCPON?	--	ZZD	1 (ON) (Note 9)	Q1

S = Sign Z = Digit with leading zeros put out as spaces D = Digit < sp > = space

NOTES:
1. Output channels 3 and 4 are not used in all modes. (See Table 5-4).
2. Current is SZD.DDDDD (0.5 A) & SZD.DDDDDD (15 mA) SZD.DDDD(2 A) & SZD.DDDDD (0.2 A)
3. Voltage can be SZD.DDDD on 7 VR
4. "X" depends upon model.
5. A space is returned for a + sign.
6. All responses are followed by a <CR> and <LF> (EOI asserted with <LF>).
7. Spaces are allowed between the header and the question mark.
8. Unit powers up with values in state register 0. Table 5-3 gives initial factory settings.
9. Factory setting. Powers up to last stored value.

Cuando se envían un conjunto de instrucciones a la fuente, ésta las ejecuta en el orden de llegada, es decir, la fuente ejecuta el comando actual antes de procesar el siguiente. Para mandar más de un comando

en la misma cadena, se usa el punto y coma, tal como se explicó en apartados anteriores. Esto maximiza la velocidad a la que la fuente acepta los comandos.

Condiciones iniciales

Cuando la fuente de alimentación sale de la fábrica, tiene almacenado en su registro de almacenamiento 0 (STO 0) la configuración mostrada en la Tabla 5.3. Al encenderse por primera vez, se hace un *autotest* y establece sus parámetros a dichos valores. Cada vez que la fuente se encienda, tomará los valores guardados en ese registro, por lo que cambiando el contenido del registro de almacenamiento 0, se podrá establecer una configuración inicial específica.

Tabla 5.3 Configuración inicial de la fuente 6622A.

Parameter	Inital Value
Voltage	0 V High Range
Current	10 mA High Range
Reprogramming Delay	20 mS
Store/Recall registers:	
0-10: Voltage	0 V High Range
Current	10 mA High Range
Over Voltage (OV)	55 V (High Range)
Over Current Protection (OCP)	Disabled
UNMASK register	0 (cleared)
SRQ	0 (Off)
Output Channels	On
Front Panel Metering	Output #1
Power Supply Address	Last stored value (Factory set to 5)
Local Control	Enabled
PON bit	On
PON SRQ	Off (0)
Cal Mode	Off

Comandos de alimentación

Se explicará brevemente cómo programar parámetros de la fuente de alimentación tales como la tensión de alimentación, corriente o los circuitos de protección.

En la fuente empleada, ambos canales tienen los mismos rangos de tensión que se pueden utilizar, por lo que los valores máximos serán los mismos y no habrá que preocuparse de la posible diferencia que hubiera entre canales. Por otro lado hay que recordar que, los comandos se emplean enviando datos en unidades en el SI.

- **Voltaje**
  Ejemplo: *VSET 1, 10*
  El canal 1 es programado a 10 V.
  Si el canal está operando en CV (*Constant voltage* o voltaje constante) entonces se programará como el voltaje actual, si por el contrario está operando en CC (*Constant current* o corriente constante) entonces se programará como el voltaje máximo de salida. Si tras definir un valor de corriente, se establece un valor de tensión dando lugar a una potencia por encima del límite, la corriente se verá reducida automáticamente. También es posible leer la tensión con la que se ha configurado el canal (*VSET? 1*) o hacer lo propio con el valor actual de voltaje (*VOUT? 1*).
- **Corriente**
  Ejemplo: *ISET 1, 0.04*
  El canal 1 es programado a 40 mA.
  Si el canal está operando en CC entonces se programará como la corriente actual, si por el contrario está operando en CV entonces se programará como la corriente máxima de salida. Si tras definir un valor de tensión, se establece un valor de corriente dando lugar a una potencia por encima del límite, la tensión se verá reducida automáticamente. También es posible leer el amperaje con el que se ha configurado el canal (*ISET? 1*) o hacer lo propio con el valor actual de corriente (*IOUT? 1*).

- **Salida *On/Off***
  Ejemplo: *OUT 1, 0; OUT 1, 1*
  El canal 1 es desactivado con un 0 y activado con un 1.
  También es posible consultar el estado actual del canal (*OUT? 1*), lo cual tiene una respuesta de 0 para desactivado y 1 para activado. Cuando está desactivado se comporta como si estuviera programado con 0 V de voltaje y 10 mA de corriente.
- **Protección de sobretensión**
  Ejemplo: *OVSET 1, 11.5*
  El canal 1 es programado con una sobretensión de 11.5 V.
  Se trata de una característica que protege a la carga de una tensión excesiva. Cuando la tensión actual supera la tensión programada, el OV actúa, poniendo la salida en baja impedancia. También es posible consultar la configuración del OV (*OVSET? 1*). Para activar la salida tras un disparo del OV, primero se debe eliminar el OV y, tras ello, resetearlo (*OVRST 1*), en caso contrario el OV actuará de nuevo.
- **Protección de sobrecorriente**
  Ejemplo: *OCP 1, 0; OCP 1, 1*
  El OCP (protección de sobrecorriente) del canal 1 es desactivado con un 0 y activado con un 1.
  Consiste en una protección la cual actúa si se detectan corrientes excesivas. También es posible consultar la configuración del OCP (*OCP? 1*), lo cual tiene una respuesta de 0 para desactivado y 1 para activado. Para activar la salida tras un disparo del OCP, se puede desactivar el OCP y mandar el comando de reseteo (*OCRST 1*) o reducir la corriente por debajo de la corriente programada y enviar dicho comando.
- **Comando de reinicio**
  Ejemplo: *CLR*
  Con este comando se devolverá a la fuente de alimentación a su estado de inicio.

5.2 Otros equipos

Además de la fuente de alimentación, serán necesarios otros equipos/elementos para llevar a cabo las pruebas que se desean realizar, éstos son:

- **Generador de señal SMU200A**
  Este equipo del fabricante Rohde & Schwarz se empleará para producir señales de RF que serán introducidas a la entrada de los dispositivos empleados en las pruebas. La interfaz disponible para manejarlo consiste en un diagrama de bloques, donde cada bloque representa una parte del generador.

  Por ejemplo, se puede observar un bloque generador en banda base, para producir señales en tiempo real u ondas arbitrarias. Es posible generar señales siguiendo un estándar de comunicaciones, señales moduladas digitalmente o definidas por el usuario. Estas señales se pueden sumar simulando entornos reales. También existen módulos para añadir a las señales en banda base efectos del canal como el *Fadding* y el *AWGN*.

  Por último, con el módulo de RF se consigue modular las señales en fase y cuadratura, seleccionar ganancias en cada rama, frecuencia, atenuadores...

Figura 5.3 Panel frontal del generador de señal SMU200A.

- **Analizador de señal N9030 PXA**
  Este analizador vectorial de Agilent es capaz de trabajar en un rango de frecuencias entre 3 Hz y 26.5 GHz, analizando señales de anchos de banda de hasta 140 MHz. Respecto al margen dinámico, diferencia del nivel entre la referencia y el ruido de fondo, es de -88 dBc para señales W-CDMA. Además, ofrece distintos modos de funcionamiento según lo que se desee realizar, éstos son: analizador de espectro, analizador IQ, modo VXA y modo VSA.

Figura 5.4 Panel frontal del analizador de señal N9030 PXA.

- ***Bias tee***
  En una de las pruebas experimentales, se emplean dos bias tee a la entrada y la salida de un transistor de efecto campo con configuración de fuente común. Mediante estas redes de tres puertos se inyecta simultáneamente la señal de RF y la polarización de DC a la puerta y al drenador del transistor.
- **Otros elementos**
  En alguna de las pruebas, además de equipos se han empleado elementos para acondicionar el camino de la salida del dispositivo hacia el analizador de señal, como por ejemplo un acoplador direccional o un atenuador. Más concretamente en otra de las pruebas experimentales de este proyecto se emplea un conjunto formado por un acoplador direccional, un atenuador y un cable coaxial para conectar la salida del dispositivo bajo prueba con el analizador de señal, que tiene unas pérdidas totales de 20.6 dB.

6 Python

Sé tú el cambio que quieres ver en el mundo.

Mahatma Gandhi

Phyton es un lenguaje de programación creado a principio de los años 90 por el neerlandés Guido van Rossum y cuyo nombre esta inspirado en el grupo de cómicos ingleses *Monty Python* del programa de la BBC *Monty Python's Flying Circus*. Se trata de un lenguaje similar a Perl[1], pero con una sintaxis muy limpia y que favorece un código legible. Es administrado por la *Python Software Foundation*[2] y es un lenguaje interpretado, con tipado dinámico, fuertemente tipado y multiplataforma.

- **Lenguaje interpretado**
  Un lenguaje interpretado significa que emplea un programa intermedio llamado intérprete para su ejecución, esto se hace realizando la traducción secuencialmente y a medida que sea necesario (paso a paso), a diferencia de los lenguajes compilados, que compilan el código completo para que posteriormente lo pueda comprender la máquina correctamente. Esto dota a Python de más flexibilidad y portabilidad, sin embargo, su ejecución es más lenta. A pesar de ello, Python tiene muchas características de los lenguajes compilados, por lo que podremos referirnos a él como un lenguaje semi interpretado. En Python, el código fuente se traduce a un pseudocódigo intermedio llamado *bytecode* generando archivos .pyc o .pyo, que son los que se ejecutarán posteriormente.
- **Tipado dinámico**
  Ésta es una de las características más potentes de los lenguajes interpretados y es que, no es necesario declarar el tipo de contenido que va a contener cada variable en su definición, sino que éste se determinará en tiempo de ejecución según el valor asignado en el programa. Incluso si, tras asignarle un valor con un tipo correspondiente, le asignamos un valor de otro tipo distinto, la variable se convertirá al tipo nuevo.
- **Fuertemente tipado**
  Es necesario realizar una conversión explícita de las variables para tratarla como un tipo distinto al que es. Es decir, si se tiene una cadena, no se podrá tratar como un número entero y sumarla con otro entero, sino que habrá que realizar una conversión previamente.
- **Multiplataforma**
  El intérprete de Python está en multitud de plataformas disponible (UNIX, Solaris, Windows, Linux, Mac OS...) por lo que si no se usan librerías específicas de la plataforma en la que se esté trabajando, el programa podrá ejecutarse en el resto de sistemas sin problemas.

[1] Lenguaje de programación desarrollado a finales de los 80 por Larry Wall con el principal objetivo de simplificar las tareas de administración de un sistema UNIX.

[2] Organización sin ánimo de lucro creada en 2001 que posee los derechos de Python y realiza tareas relacionadas con dicho lenguaje tales como el desarrollo de Python o la administración las licencias de código abierto del mismo.

- **Programación Orientada a Objetos**
  La POO (Programación Orientada a Objetos) es un paradigma de programación[3] en el cual los conceptos e ideas reales de los problemas a solucionar se tratan como objetos y clases en el programa, el cual consistirá en una interacción entre los mismos. Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.

6.1 Herramientas básicas

Existen dos formas de ejecutar código Python. Se pueden escribir líneas en el intérprete y obtener una respuesta para cada una de ellas, lo que se conoce como modo interactivo, o bien hacerlo en un archivo de texto y ejecutarlo.

Modo interactivo

Se dice que se está trabajando en una sesión interactiva cuando los comandos son leídos directamente de la terminal. Normalmente, se espera el siguiente comando con el *prompt primario* (">>>") y para las líneas de continuación se espera con el *prompt secundario*. El intérprete, antes de mostrar el prompt primario, muestra un mensaje con su número de versión y una nota de copyright como se muestra en el Código 6.1.

Código 6.1 Inicio de sesión interactiva en Python.

```
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 03:27:45) [MSC v.1900 64
    bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Si se quiere escribir un constructor multilínea, será necesario usar las líneas de continuación como en el Código 6.2.

Código 6.2 Ejemplo de multilínea en una sesión interactiva.

```
>>> flag = 1
>>> if flag:
... print ("La bandera está activa")
...
La bandera está activa
```

En este proyecto nos interesa escribir programas por lo que no se empleará el intérprete directamente como se ha expuesto anteriormente. Sin embargo, indirectamente sí que se está usando ya que al ejecutar los programas, en realidad lo que se está haciendo es ejecutar línea a línea en la terminal.

Anaconda Navigator

Como se ha nombrado previamente, también es posible escribir archivos en los que se incluya todo el código del programa para posteriormente ejecutarlos. En este proyecto se realizará de esta manera, empleando Anaconda Navigator, que es una GUI que permite iniciar aplicaciones y administrar y manejar fácilmente paquetes y entornos sin utilizar líneas de comandos. Con esta interfaz, se puede trabajar con un repositorio local o en la llamada *Anaconda Cloud*. Está disponible para macOS, Linux y Windows. La forma más simple de ejecutar el código con Navigator es con Spyder (*Scientific PYthon Development EnviRonment*). Haciendo click en Spyder desde el inicio de Navigator se abrirá una nueva pestaña (Figura 6.2) donde se podrá escribir y ejecutar el código.

Spyder es un potente entorno de desarrollo interactivo para el lenguaje Python con funciones avanzadas de edición, pruebas interactivas y depuración.

[3] Se trata de una representación de un enfoque particular para diseñar soluciones a un problema. Cada paradigma tiene sus propios conceptos y formas de tratar los elementos involucrados en el problema, así como los pasos a seguir para solucionarlo.

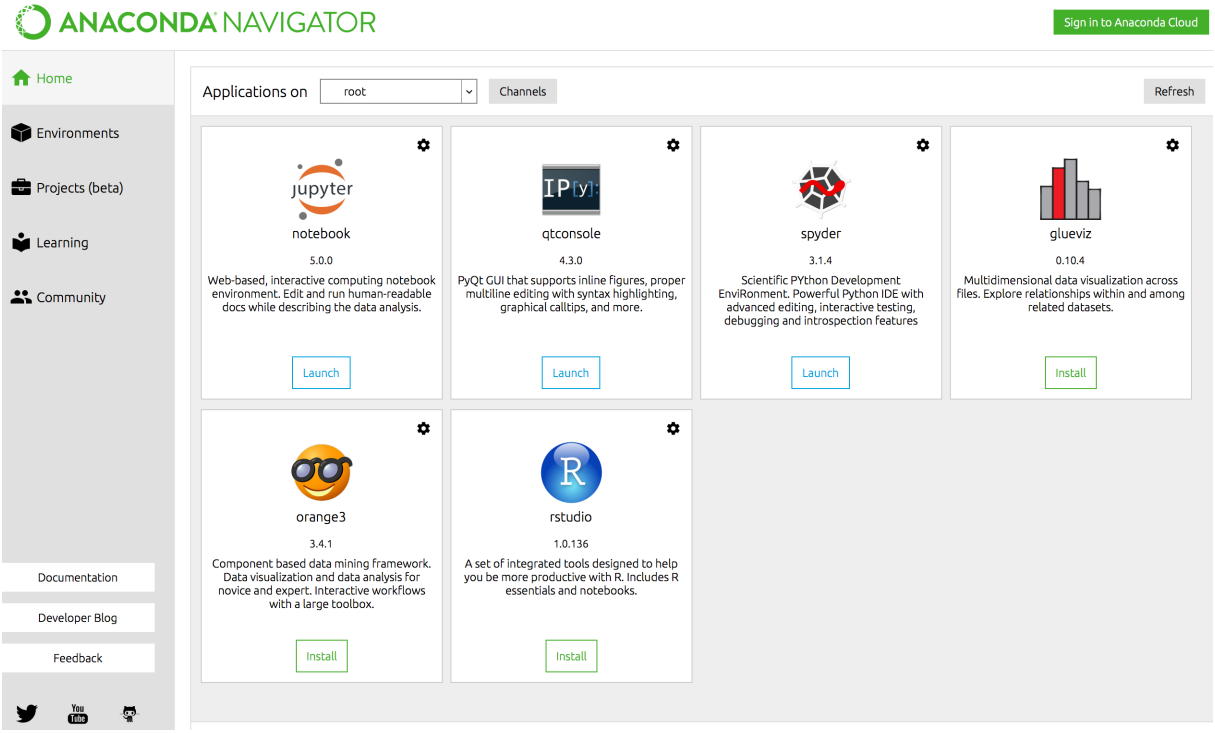


Figura 6.1 Inicio Anaconda Navigator.

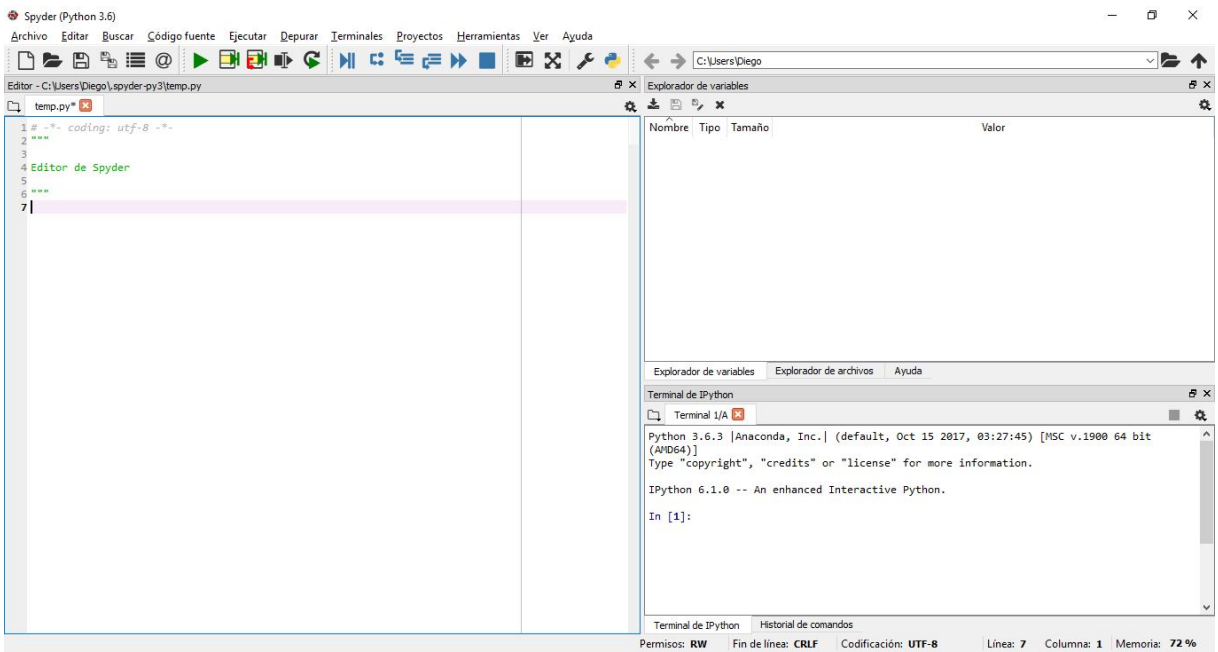


Figura 6.2 Ventana de Spyder.

6.2 Palabras reservadas

A las palabras que son identificadores predefinidos y que tienen un significado especial dentro de cada lenguaje, y, por lo tanto, no se pueden utilizar como identificadores en los programas, se les llama palabras reservadas. En Python, comparado con otros lenguajes, el número de palabras reservadas no es muy alto, pero habrá que tenerlas en cuenta y son las mostradas en la Tabla 6.1.

Tabla 6.1 Palabras reservadas en Python.

and	class	elif	finally	if	lambda	print	while
as	continue	else	for	import	not	raise	with
assert	def	except	from	in	or	return	yield
break	del	exec	global	is	pass	try	

6.3 Variables en Python

En Python las variables se entienden como "etiquetas" que permiten referenciar a los datos los cuales se guardan en objetos. Al ser un lenguaje de programación orientado a objetos, su modelo de datos también está basado en ello. Para cada dato que aparece, Python crea un objeto para guardarlo, y cada uno de ellos tiene:

- Identificador único: permite a Python referirse a él sin ambigüedades.
- Tipo de datos: permite a Python saber qué operaciones pueden hacerse con el dato y como almacenarlo en memoria.
- Valor: el dato en cuestión.

Por ejemplo al escribir $a=2$, Python crea el objeto "2" el cual tiene un identificador único y es de tipo entero, guardando el valor "2". Posteriormente asignará la etiqueta "a" a dicho objeto.

Por otro lado los objetos pueden ser:

Objetos inmutables: Objetos que no se pueden modificar (números, cadenas y tuplas).

Objetos mutables: Objetos que se pueden modificar (listas y diccionarios).

A continuación se muestran unos ejemplos acerca de estos dos tipos de objetos, ya que es algo importante a la hora de programar en Python:

Código 6.3 Cambio de valor de una variable (objetos inmutables).

```
>>> x = 15
>>> y = x
>>> y, x
(15, 15)
>>> x = 30
>>> x, y
(30, 15)
```

En el Código 6.3 se ve cómo se crea un nuevo objeto "30" y se referencia con la etiqueta "x" mientras que "y" sigue referenciando al objeto antiguo.

Código 6.4 Cambio de valor de una variable (objetos mutables que no se modifican).

```
>>> x = [15, 30]
>>> y = x
>>> y, x
([15, 30], [15, 30])
```

```
>>> x = [25, 40]
>>> x, y
([25, 40], [15, 30])
```

En el Código 6.4 ocurre igual que en el caso anterior pero con otro tipo de estructura. Se crea un objeto nuevo referenciándolo con la etiqueta "x" e "y" sigue referenciando al antiguo.

Código 6.5 Cambio de valor de una variable (objetos mutables que sí se modifican).

```
>>> x = [15, 30]
>>> y = x
>>> x, y
([15, 30], [15, 30])
>>> x[0] = 25
>>> x, y
([25, 30], [25, 30])
```

En este último Código 6.4, a diferencia de los anteriores, no se crea ningún objeto nuevo sino que se modifica el existente. Como ambas etiquetas están referenciando al mismo objeto, devuelven el mismo valor modificado.

6.4 Tipos de datos

6.4.1 Números y operaciones aritméticas elementales

Enteros y decimales

Python distingue entre números enteros y decimales. Para denotar la diferencia entre uno y otro se usará el delimitador punto ("."). Los primeros se almacenan en variables de tipo *int* y los segundos en variables de tipo *float*.

Código 6.6 Números enteros y decimales.

```
>>> type(4)
<class 'int'>
>>> type(4.0)
<class 'float'>
```

Números complejos

También es posible almacenar números complejos en Python y hacer cálculos con ellos. Para crear una variable de este tipo se puede hacer de dos maneras distintas: usando *complex* o escribiendo directamente la parte real y la parte imaginaria acompañada del carácter "j", las cuales se pueden mostrar por separado ya que cada una se guarda en un atributo distinto del objeto creado, tal y como se muestra en el Código 6.7.

Código 6.7 Números complejos.

```
>>> n_complejo1 = complex(1,6)
>>> n_complejo1
(1+6j)
>>> n_complejo2 = 1 + 6j
>>> n_complejo2
(1+6j)
>>> n_complejo2.real
1.0
>>> n_complejo2.imag
6.0
```

Operaciones básicas

Las operaciones básicas son la suma ("+"), resta ("-"), multiplicación ("*") y división ("/"). En cualquiera de las operaciones, si interviene algún número decimal, el resultado será decimal. En el caso de la división, incluso aunque no intervenga ningún número decimal y el resultado sea un número entero, se guardará en un decimal. En el Código 6.8 se muestran algunos ejemplos de esto.

Código 6.8 Operaciones básicas.

```
>>> 5 + 4
9
>>> 2 - 5
-3
>>> 10 / 2
5.0
>>> 3.4 * 5
17.0
```

Cociente y resto de una división

El cociente siempre será un número entero (su parte decimal será "0") aunque se almacenará en un decimal si interviene algún decimal en la operación, sin embargo, el resto puede ser decimal. Este último siempre tiene el mismo signo que el divisor. En el Código 6.9 se pone en práctica cómo hallar el cociente y el resto de una división con los operadores ("//" y "%") o la función correspondiente.

Código 6.9 Cociente y resto de una división.

```
>>> 16.5 // 2
8.0
>>> 16.5 % 2
0.5
>>> divmod(16.5,2)
(8.0, 0.5)
```

Potencias y raíces

Las potencias tienen prioridad sobre las multiplicaciones y divisiones y, teniendo en cuenta que se tratan de operaciones inversas, se empleará el mismo operador ("**") para calcular las raíces. También se puede realizar esto mediante una función.

Código 6.10 Potencias y raíces.

```
>>> 2 ** 5
32
>>> 16 ** 0.5 # Halla la raíz cuadrada
4.0
>>> (-1) ** 0.5
(6.123233995736766e-17+1j)
>>> -9 ** 0.5  # Primero hace la raíz y luego le pone el signo negativo
-3.0
>>> pow(4,3)
64
>>> pow(64,0.5)
8.0
```

Otras funciones de interés

round: Si recibe un argumento, redondea al entero más próximo, si recibe dos, redondea al decimal más proximo con el número de decimales indicado en el segundo argumento.

floor: Redondea al entero inferior.

ceil: Redondea al entero superior.

abs: Calcula el valor absoluto de un número.

max: Calcula el valor máximo de un conjunto de valores (numéricos o cadenas). En el caso de cadenas, el valor máximo se corresponde con el último valor en orden alfabético.

min: Calcula el valor mínimo de un conjunto de valores (numéricos o cadenas). En el caso de cadenas, el valor mínimo se corresponde con el primer valor en orden alfabético.

sum: Calcula la suma de un conjunto de valores contenidos en una variable que debe ser iterable.

sorted: Ordena un conjunto de valores contenidos en una variable que debe ser iterable.

6.4.2 Cadenas

Una cadena es una secuencia inmutable de caracteres Unicode[4], delimitada por comillas. Las cadenas tienen mucha utilidad en la programación y se realizará un breve análisis de las funciones más relevantes. Antes que nada, destacar que, como en muchos lenguajes de programación, para acceder al primer valor de una cadena se accede con el índice "0" y no con el "1" como sería lógico pensar para alguien externo al mundo de la programación:

Código 6.11 Índice de las cadenas.

```
>>> cadena = "prueba"
>>> cadena[0]
'p'
>>> cadena[1]
'r'
```

Longitud

Para obtener la longitud de una cadena, se hará con la función *len*:

Código 6.12 Longitud de cadenas.

```
>>> cadena = "prueba"
>>> len(cadena)
6
```

Concatenar cadenas

Es una de las propiedades más importantes, con la que se consigue juntar varias cadenas en una sola. Es posible aplicar operadores a las cadenas como se muestra en el Código 6.13.

Código 6.13 Concatenación de cadenas.

```
>>> cadena = "prueba"
>>> "mensaje" + "DE" + "prueba"
'mensajeDEprueba'
>>> "mensaje" + "DE" + 4*cadena
'mensajeDEpruebapruebapruebaprueba'
```

Porciones de cadenas

A las subcadenas se les llama porciones de cadenas y su acceso se hará similar al que se indicó anteriormente para acceder a un carácter. Con el operador "[n:m]" se extrae desde el carácter n-ésimo hasta el m-ésimo.

[4] Se trata de un estándar de codificación de caractéres universal para representar texto de múltiples lenguajes.

Código 6.14 Porciones de cadenas.

```
>>> cadena = "prueba"
>>> cadena[2:4]
'ue'
>>> cadena[:3]
'pru'
>>> cadena[1:]
'rueba'
```

Cadenas "f"

En Python 3.6 se incluyó una facilidad para insertar variables y expresiones dentro de las cadenas, lo cual se denominó cadenas "f". Éstas contienen variables y expresiones entre llaves las cuales se sustituyen por su valor. Para emplearlas se debe anteponer una "f" a la cadena como se indica en el Código 6.15.

Código 6.15 Cadenas "f".

```
>>> mes = "Marzo"
>>> dias = 31
>>> print (f"El mes de {mes} tiene {dias} días.")
El mes de Marzo tiene 31 días.
```

6.4.3 Booleanos

Son aquellas variables que solo pueden tomar dos posibles valores: verdadero (*True*) o falso (*False*). En realidad, en Python todas las variables pueden considerarse booleanas, ya que una variable vacía o nula se supone *False* y el resto *True*. Para convertir un valor a booleano se emplea la funcion *bool*.

Código 6.16 Variables booleanas.

```
>>> bool()
False
>>> bool(None)
False
>>> bool(111)
True
>>> bool("ejemplo")
True
```

Operadores lógicos

Se tratan de operaciones que trabajan con operadores booleanos y son los que se muestran en la Tabla 6.2.

Tabla 6.2 Operadores lógicos en Python.

Operador	Descripción
and	"Y" lógico
or	"O" lógico
not	Negación

Estos operadores se pueden agrupar en una sola expresión, dando lugar a expresiones compuestas, sin embargo, hay que hacerlo teniendo en cuenta que Python evalúa primero los *not*, luego los *and* y por último los *or*.

Operadores relacionales

También operan con resultados booleanos.

Tabla 6.3 Operadores relacionales en Python.

Operador	Descripción
>	Mayor que
<	Menor que
>=	Mayor o igual que
<=	Menor o igual que
==	Igual que
!=	Distinto que

Pyhon permite encadenar varias comparaciones y el resultado será verdadero si y sólo si todas las comparaciones lo son.

6.4.4 Secuencias

Tuplas

Es un conjunto ordenado de elementos que pueden ser del mismo o de varios tipos. Se consideran objetos inmutables. Para crearla solo hay que escribir los elementos entre paréntesis (los cuales son opcionales) separados por comas. El acceso a los elementos se realiza introduciendo el índice correspondiente entre corchetes y para saber el número de elementos que componen una tupla se emplea la funcion *len*. Para crear tuplas de un solo elemento es necesario escribir una coma tras él. En el Código 6.17 se ejemplifica todo lo previamente explicado.

Código 6.17 Ejemplo de uso de tuplas.

```
>>> tupla_ejemplo = (3, "hola", 9.2)
>>> tupla_ejemplo[2]
9.2
>>> len(tupla_ejemplo)
3
>>> tupla_un_elemento = (10,)
>>> tupla_un_elemento
(10,)
```

Listas

Son conjuntos de elementos del mismo o distinto tipo, las cuales se limitan por corchetes. Se consideran objetos mutables. Una lista puede contener incluso otras listas y éstas a su vez a otras, y así sucesivamente. Para acceder a algún elemento de la lista, se realiza de la misma manera que en las tuplas, con los corchetes e introduciendo su índice. Si aplicamos el operador suma con listas, éstas se concatenan y para eliminar elementos de las mismas, se hará con la palabra reservada *del*.

Por último, comentar que si se hace una copia de una lista, al ser un objeto mutable, si se modifica la lista inicial posteriormente, la nueva también se verá modificada, por lo que se empleará la notación de sublistas. Todo esto se puede observar en el Código 6.18.

Código 6.18 Ejemplo de uso de listas.

```
>>> ejemplo_lista = [1, 2, 3, 4, 5, 6]
>>> ejemplo_lista[0]
1
>>> ejemplo_lista_anidada = [ejemplo_lista, 7, 8]
>>> ejemplo_lista_anidada
```

```
[[1, 2, 3, 4, 5, 6], 7, 8]
>>> ejemplo_lista + [3, 4]
[1, 2, 3, 4, 5, 6, 3, 4]
>>> del ejemplo_lista[0]
>>> ejemplo_lista
[2, 3, 4, 5, 6]
>>> ejemplo_lista2 = ejemplo_lista[:] # Notación de sublistas
>>> ejemplo_lista[4] = 10
>>> ejemplo_lista
[2, 3, 4, 5, 10]
>>> ejemplo_lista2
[2, 3, 4, 5, 6]
>>> ejemplo_lista3 = ejemplo_lista2  # Sin notación de sublistas
>>> ejemplo_lista2[4]=20
>>> ejemplo_lista2
[2, 3, 4, 5, 20]
>>> ejemplo_lista3
[2, 3, 4, 5, 20]
```

Range

Es una lista inmutable, es decir, no se puede modificar, de números enteros en sucesión aritmética. Se puede llamar con uno, dos o tres argumentos:

- Un argumento (n): la lista va desde 0 hasta el n-1 de uno en uno.
- Dos argumentos (m, n): la lista va desde m hasta n-1 de uno en uno.
- Tres argumentos (m, n, p): la lista va desde m hasta justo antes de superar o igualar a n de p en p.

Para ver los valores del range(), es necesario convertirlo a lista mediante la función *list*.

Código 6.19 Ejemplo de uso de range.

```
>>> ejemplo_range = range(5)
>>> ejemplo_range
range(0, 5)
>>> list(ejemplo_range)
[0, 1, 2, 3, 4]
```

Diccionarios

Esta estructura permite al programador almacenar cualquier tipo de valor en ella, además, tienen la particularidad de que cada elemento se puede relacionar con una clave para acceder a él, la cual debe ser inmutable. Esta clave, al igual que se hacía en las tuplas y en las listas con el índice, se introduce entre corchetes. Para crearlos se emplean las llaves.

Código 6.20 Ejemplo de uso de diccionarios.

```
>>> ejemplo_diccionario = {'primero': 1, 'segundo': 2, 'tercero': 3}
>>> ejemplo_diccionario
{'primero': 1, 'segundo': 2, 'tercero': 3}
>>> ejemplo_diccionario['primero']
1
```

6.5 Estructuras de control

6.5.1 Condicionales

if...

Permite ejecutar ciertas instrucciones cuando se cumpla una condición. La sintaxis es la siguiente:

if condición:
    órdenes a ejecutar si se cumple la condición

Como se puede observar, la primera línea contiene la condición terminada en el carácter ":" que se evalúa devolviendo un valor lógico, si el resultado es *True* se ejecuta el bloque de órdenes. Es importante destacar que dicho bloque debe tener un sangrado para indicar las instrucciones que pertenecen al mismo.

Código 6.21 Ejemplo práctico de la sentencia if.

```
>>> numero = 10
>>> if numero == 10:
...   print ("El número es correcto.")
...
El número es correcto.
```

if... else...

Permite ejecutar ciertas instrucciones cuando se cumpla una condición y otras distintas en caso contrario. La sintaxis es la siguiente:

if condición:
    órdenes a ejecutar si se cumple la condición
else:
    órdenes a ejecutar si no se cumple la condición

En este caso, se cumple lo mismo que en la anterior, sin embargo, en caso de que al evaluar la condición ésta devuelva un valor *False*, se ejecutará el segundo bloque de órdenes.

Código 6.22 Ejemplo práctico de la sentencia if-else.

```
>>> numero = 20
>>> if numero == 10:
...   print ("El número es correcto.")
... else:
...   print ("El número es incorrecto.")
...
El número es incorrecto
```

6.5.2 Iteraciones

for

Los bucles *for* son una estructura las cuales repiten un bloque de instrucciones un número de veces predeterminado. La sintaxis es la siguiente:

for variable **in** elemento interable:
    cuerpo del bucle

El elemento interable puede ser una lista, cadena... El cuerpo del bucle se ejecutará tantas veces como elementos tenga dicho iterable.

Código 6.23 Ejemplo práctico de la sentencia for.

```
>>> for i in range(4):
... print (i)
...
0
1
2
3
```

while

Los bucles *while* permiten que, mientras una cierta expresión (compuesta por una o varias condiciones) se cumpla, se ejecute un bloque de instrucciones. La sintaxis es la siguiente:

while condición:
    cuerpo del bucle

Una vez que se llega a un bucle *while* se evalúa la condición, si ésta devuelve un valor *True*, el cuerpo se ejecutará una vez y una vez terminado, se evaluará de nuevo la condición. Este proceso se hará indefinidamente hasta que la la condición sea *False*. Para que ésto ocurra, como es lógico, el valor de la (o las) variables de control, que son las incluidas en la condición de entrada, deben cambiar dentro del bucle.

Código 6.24 Ejemplo práctico de la sentencia while.

```
>>> while i <= 6:
...   print (i**2)
...   i+=1
...
1
4
9
16
25
36
```

6.6 Funciones

Las funciones en Python pueden escribirse en cualquier punto de un programa definiéndolas correctamente, aunque es una buena práctica hacerlo al inicio del mismo. Para hacerlo será necesario incluir la palabra reservada *def*, el nombre de la función y los argumentos que reciba en caso de ser necesarios. Otro elemento opcional es incluir un *return*.

Código 6.25 Ejemplo práctico de función.

```
>>> def mi_funcion(numero):
...   print ("Esta función halla la raíz cuadrada de un número.")
...   print (numero ** 0.5)
...
>>> mi_funcion(9)
Esta función halla la raíz cuadrada de un número.
3.0
```

6.6.1 Variables

Uno de los principales problemas de las funciones es el conflicto de nombres de variables ya que si pegamos en un programa una subrutina que emplee alguna variable con el mismo nombre que una ya existente, podría

provocarse un comportamiento no deseado del programa. Es por ello que existe el alcance de las variables y se resume en:

1. Cada variable pertenece a un ámbito determinado: al programa principal o a una subrutina.
2. Las variables son inaccesibles en un ámbito superior al que pertenecen pero pueden serlo en ámbitos inferiores.
3. En Python hay tres tipos: las que pertenecen a la subrutina (**locales**), las que pertenecen al programa principal (**globales**) y las que pertenecen a un ámbito superior al de la subrutina pero no son globales (**no locales**).

6.6.2 Parámetros

Los parámetros, que son los argumentos que espera una función recibir al ser invocada, en Python pueden recibirse de varias maneras:

1. **Por omisión**
   La función tiene algún parámetro predefinido, por lo que no es necesario darle valor al llamarla.
2. ***Keywords***
   Los parámetros son dados como pares de *clave*=*valor*, pudiendo no hacerlo en el mismo orden al que tenga la función en su definición.
3. **Parámetros arbitrarios**
   Es posible que una función reciba un número de parámetros arbitrarios (desconocido) en cada invocación y se recibirán en forma de tupla. Para definirlo, se antecede el parámetro con un asterisco. También es posible recibir parámetros arbitrarios en forma de pares *clave*=*valor* lo cual se indicará con dos asteriscos. Si una función espera recibir parámetros fijos y arbitrarios, los arbitrarios siempre deben suceder a los fijos.
4. **Desempaquetado**
   En este caso, se produce una situación inversa a la anterior. La función recibe un número fijo de parámetros pero al ser invocada, se le pasará como argumento una lista o tupla (precedida por un asterisco) o un diccionario (precedido por dos asteriscos).

6.7 Entrada/salida

A la hora de realizar el proyecto, será necesario que el usuario interactúe con el programa realizado introduciendo o leyendo datos, por lo que este aspecto será importante a la hora de programar.

6.7.1 Entrada

Se trata de los datos que llegan al programa desde el exterior. Se realizará con la función *input*.

Esta función permite obtener texto del teclado. El programa, al llegar a la función, espera que se introduzca algo por teclado.

6.7.2 Salida

Son los datos que el programa proporciona al exterior. Usualmente la vía de salida es la pantalla, para lo cual se usará la función *print*.

Dicha función permite mostrar texto en pantalla, el cual será pasado por argumento sucedido por un salto de línea, el cual podrá ser sustituido si se le pasa otra cadena como segundo argumento precedida de *end*=.

6.7.3 Ficheros

Para trabajar con ficheros se emplearán las palabras reservadas *with* y *as* junto a la función *open*. La sintaxis es la siguiente:

```
with open("FICHERO", mode="MODO", encoding="CODIFICACIÓN") as
fichero:
    bloque de instrucciones
```

Donde, como primer argumento a la función se le pasará la ruta absoluta del fichero, como segundo el modo de apertura (Tabla 6.4) y como tercero el juego de carácteres del mismo.

Tabla 6.4 Modos de apertura de un archivo.

Indicador	Modo de apertura	Ubicación del puntero
r	Solo lectura	Al inicio del archivo
rb	Solo lectura en modo binario	Al inicio del archivo
r+	Lectura y escritura	Al inicio del archivo
rb+	Lectura y escritura en modo binario	Al inicio del archivo
w	Solo escritura. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb	Solo escritura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
w+	Escritura y lectura. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
wb+	Escritura y lectura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe	Al inicio del archivo
a	Añadido (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab	Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
a+	Añadido (agregar contenido) y lectura. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo
ab+	Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo

6.8 POO: Programación Orientada a Objetos

Los elementos de la POO pueden entenderse como los elementos que se necesitan para diseñar y programar un sistema. Los principales se explicarán en esta sección. Por otro lado las características se entienden como las herramientas para construir dicho sistema.

6.8.1 Elementos principales

Clases

Son los modelos sobre los cuales se construirán los objetos. En Python se define con la palabra *class* seguida del nombre.

Propiedades

Se tratan de las características las cuales tiene el objeto. Se representan a modo de variables dentro de cada clase.

Métodos

Son funciones y consisten en las acciones propias que puede realizar el objeto en concreto.

Objetos

Los objetos son la manera de materializar las clases. A dicha acción se le denomina instanciar una clase y consiste en asignar la clase como valor a una variable.

6.8.2 Herencia

Algunos objetos comparten propiedades y métodos, además de añadir nuevos. A esto se le llama heredar de una clase. Es importante destacar que en Python, cuando una clase no hereda de ninguna otra, debe hacerse heredar de object, que es la clase principal de Python, que define un objeto.

6.9 Módulos y paquetes

En Python, los módulos son los archivos con extensión *.py*. Un conjunto de módulos puede formar lo que se denomina como paquete, que consiste en una carpeta que contiene varios módulos y un archivo de inicio llamado *__init__.py*, el cual puede estar vacío pero debe existir. Para importar módulos se hará con la palabra reservada *import* como se observa en el Código 8.1.

Código 6.26 Importación de módulos.

```
# -*- coding: utf-8 *-*

import modulo       # Importa un módulo que no pertenece a un paquete
import paquete.modulo1 # Importa un módulo que está dentro de un paquete
import paquete.subpaquete.modulo1
```

6.9.1 Namespaces

Para acceder a un elemento de un módulo importado, se emplea el *namespace* seguido de un punto y el nombre del elemento en cuestión.

Código 6.27 Uso de namespaces.

```
print modulo.CONSTANTE_1
print paquete.modulo1.CONSTANTE_1
print paquete.subpaquete.modulo1.CONSTANTE_1
```

Por otro lado, esto se puede hacer empleando un alias en lugar del propio *namespace*, para ello, se debe asignar el alias durante la importación como se indica en el Código 6.28. Gracias a esto, no será necesario escribir el módulo cada vez que se quiera acceder a dicho elemento ya que se hará más fácilmente mediante el alias.

Código 6.28 Asignación de alias.

```
import modulo as m
import paquete.modulo1 as pm
import paquete.subpaquete.modulo1 as psm

print m.CONSTANTE_1
print pm.CONSTANTE_1
print psm.CONSTANTE_1
```

6.9.2 Módulos de sistema

Son los que integran la librería estándar de Python. Los más destacables son los módulos *os*, *sys* y *subprocess* que se explicarán a continuación.

Módulo os

Permite acceder a funcionalidades que dependen del Sistema Operativo, las cuales nos ofrecen principalmente información sobre el entorno y manipular la estructura de directorios. Los métodos más destacables se muestran en la Tabla 6.5

Tabla 6.5 Métodos módulo os.

Método	Descripción
os.access(path, modo_de_acceso)	Saber si se puede acceder a un archivo o directorio
os.getcwd()	Conocer el directorio actual
os.chdir(nuevo_path)	Cambiar de directorio de trabajo
os.chroot()	Cambiar al directorio de trabajo raíz
os.chmod(path, permisos)	Cambiar los permisos de un archivo o directorio
os.chown(path, permisos)	Cambiar el propietario de un archivo o directorio
os.mkdir(path[, modo])	Crear un directorio
os.mkdirs(path[, modo])	Crear directorios recursivamente
os.remove(path)	Eliminar un archivo
os.rmdir(path)	Eliminar un directorio
os.removedirs(path)	Eliminar directorios recursivamente
os.rename(actual, nuevo)	Renombrar un archivo
os.symlink(path, nombre_destino)	Crear un enlace simbólico

Módulo sys

Es el encargado de proporcionar funcionalidades y variables relacionadas con el intérprete. Las más importantes se resumen en la Tabla 6.6 y en la Tabla 6.7.

Tabla 6.6 Variables del módulo sys.

Variable	Descripción
sys.argv	Retorna una lista con todos los argumentos pasados por línea de comandos. Al ejecutar python modulo.py arg1 arg2, retornará una lista: ['modulo.py', árg1', árg2']
sys.executable	Retorna el path absoluto del binario ejecutable del intérprete de Python
sys.maxint	Retorna el número positivo entero mayor, soportado por Python
sys.platform	Retorna la plataforma sobre la cuál se está ejecutando el intérprete
sys.version	Retorna el número de versión de Python con información adicional

Tabla 6.7 Métodos del módulo sys.

Método	Descripción
sys.exit()	Forzar la salida del intérprete
sys.getdefaultencoding()	Retorna la codificación de caracteres por defecto
sys.getfilesystemencoding()	Retorna la codificación de caracteres que se utiliza para convertir los nombres de archivos unicode en nombres de archivos del sistema
sys.getsizeof(object[, default])	Retorna el tamaño del objeto pasado como parámetro. El segundo argumento (opcional) es retornado cuando el objeto no devuelve nada.

Módulo subprocess

Permite trabajar de forma directa con órdenes del Sistema Operativo. Hay que tener cuidado ya que el uso incorrecto de este módulo podría comprometer al sistema.

Entre los métodos más comunes de *subprocess*, se encuentra *call*. Este método suele ser útil para ejecutar órdenes sencillas. El método *call*, recibe como argumento el comando a ser ejecutado. Sin embargo, si el comando introducido requiere algún argumento, se le pasará a call como primer parámetro el comando y el resto los argumentos.

6.10 Gráficas en Python

Para realizar gráficas, se utiliza Pylab. Pylab es una API para Python de la biblioteca gráfica Matplotlib (matplotlib.pyplot) que utiliza Numpy, un módulo matemático que añade funciones para operar con vectores

o matrices. Esta biblioteca proporciona una forma rápida de representar datos y figuras con calidad y en varios formatos.

6.10.1 Funciones principales

En este apartado se nombrarán algunas de las funciones que tiene el módulo para hacer gráficas, ya que sería muy extenso si se explicaran todas.

Crear figuras

Crear una nueva figura se realiza de la siguiente manera:

```
figure(num, figsize, dpi, facecolor, edgecolor, frameon)
```

Los argumentos, los cuales son opcionales, son:

1. num: Numeración de la figura.
2. figsize: Tupla que contiene el ancho y alto en pulgadas del tamaño de la figura
3. dpi: Resolución de la imagen en puntos por pulgada.
4. facecolor: Color del rectángulo de la figura.
5. edgecolor: Color del perímetro de la figura.
6. frameon:Si es falso, elimina el marco de la figura.

Múltiples gráficas en una figura

Para organizar varias gráficas en la misma cuadrícula, se sigue la siguiente estructura:

```
subplot(numRows, numCols, plotNum)
```

Los argumentos son:

1. numRows: Número de filas.
2. numCols: Número de columnas.
3. plotNum: Número de gráfica.

Pintar gráfica

Finalmente, la sintaxis a seguir para pintar una gráfica es la siguiente:

```
subplot(numRows, numCols, plotNum)
```

Los argumentos son:

1. x: Valor en el eje de abcisas (Puede ser tupla, lista o array).
2. y: Valor en el eje de ordenadas (Puede ser tupla, lista o array).
3. linestyle: Color y tipo de gráfica.
4. linewidth: Ancho de línea.
5. marker: Marcador.

Otras funciones

Otras de las funciones útiles y que se emplearán en el proyecto son:

show: Presenta las figuras en pantalla.

legends: Coloca una leyenda a las gráficas

xlabel: Etiqueta al eje de abcisas de la gráfica actual.

ylabel: Etiqueta al eje de ordenadas de la gráfica actual.

close: Cierra la gráfica.

Ejemplos de gráficas

Para empezar, en el Código 6.29 se muestra cómo pintar una gráfica en una dimensión, la cual se puede observar en la Figura 6.3.

Código 6.29 Gráfica simple en una dimensión.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Carga de los módulos necesarios
import scipy as sp
import matplotlib.pyplot as plt

# Creamos el array x de cero a diez con cien puntos
x = sp.linspace(0, 10, 100)

# Creamos el array y conde cada punto es el seno de cada elemento de x
y = sp.sin(x)

# Creamos una figura
plt.figure()

# Representamos
plt.plot(x,y)

# Mostramos en pantalla
plt.show()
```

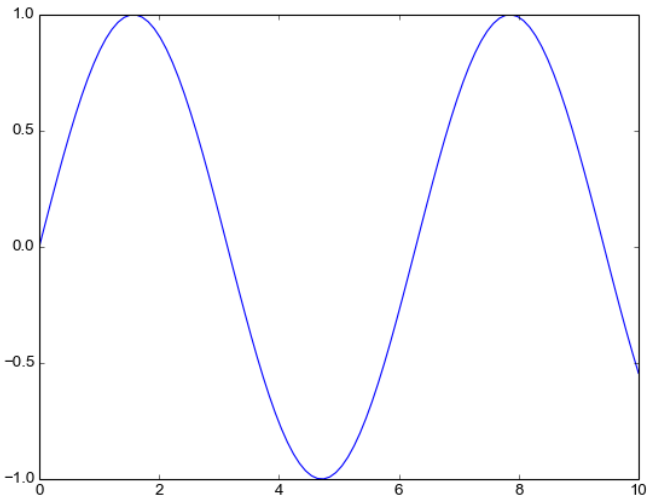


Figura 6.3 Gráfica simple en una dimensión.

A continuación, en el Código 6.30 se muestra cómo programar para que se muestren varias gráficas en la misma figura, como se puede observar en la Figura 6.4.

Código 6.30 Varias gráficas individuales en una figura.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import division

import numpy as np               # Cargamos numpy como el alias np
import matplotlib.pyplot as plt  # Cargamos matplotlib.pyplot como el alias plt

# Definimos el periodo de la gráfica senoidal
periodo = 2

# Definimos el array dimensional
x = np.linspace(0, 10, 1000)
# Definimos la función senoidal
y = np.sin(2*np.pi*x/periodo)

# Creamos la figura
plt.figure()

# Primera gráfica
plt.subplot(2,2,1)
plt.plot(x, y,'r')

 # Segunda gráfica
plt.subplot(2,2,2)
plt.plot(x, y,'g')

# Tercera gráfica
plt.subplot(2,2,3)
plt.plot(x, y,'b')

# Cuarta gráfica
plt.subplot(2,2,4)
plt.plot(x, y,'k')

# Mostramos en pantalla
plt.show()
```

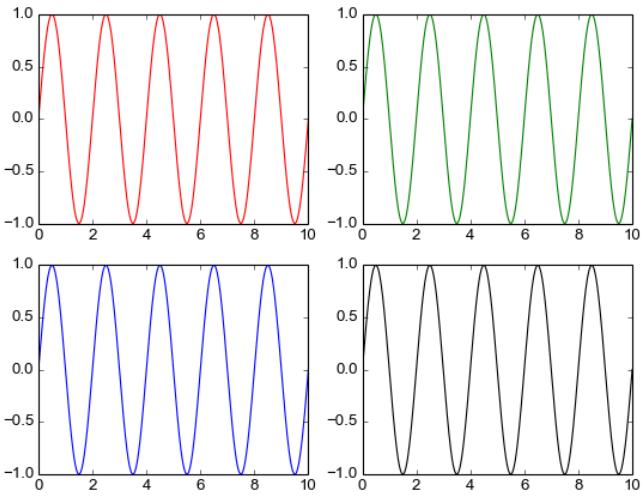


Figura 6.4 Varias gráficas individuales en una figura.

6.11 PyVISA

Se trata de un paquete de Python que permite el control remoto de dispositivos independientemente del bus empleado. Esto recuerda al estándar VISA visto en la Sección 2.2 ya que PyVISA no es más que una forma sencilla de implementar dicho estándar en Python. De esta manera se facilitarán las tareas de monitorización y control de instrumentos, ya que esta librería se podrá emplear con cualquier interfaz y con cualquier instrumento.

6.11.1 Control remoto paso a paso

Para realizar una comunicación con un instrumento y, por tanto, poder enviarle y recibir información, habrá que realizar una serie de pasos previos que se explicarán a continuación:

Creación del objeto correspondiente al instrumento

Lo primero que hay que realizar para establecer una comunicación, es crear el objeto asociado al instrumento con el que se quiere realizar la misma. Para ello, la estructura será la siguiente:

```
import visa
resources = visa.ResourceManager()
IV_ Obj = resources.open_ resource('interface_ type::board_ number::resource_ class')
```

En la primera línea se importa el módulo VISA para emplearlo posteriormente. En la segunda línea se llama al método *ResourceManager* de dicho módulo, con esto se consigue guardar en una variable los recursos existentes. Para finalizar, en la última línea se abre el recurso que se requiera de la lista previa. Los campos en cursiva habrá que rellenarlos de la siguiente manera:

interface_ type: Tipo de interfaz del recurso.

board_ number: Número de la tarjeta de la interfaz del recurso.

resource_ class: Especifica la clase de recurso.

En el Código 6.31 se puede ver un ejemplo de la creación de un objeto GPIB:

Código 6.31 Creación del objeto GPIB.

```
>>> import visa
>>> resources = visa.ResourceManager()
>>> resources.list_resources()
('ASRL1::INSTR', 'ASRL2::INSTR', 'GPIB0::14::INSTR')
>>> IV_Obj = resources.open_resource('GPIB0::14::INSTR')
```

Establecimiento de los atributos del objeto

Un recurso representa un instrumento. Existen muchas clases que derivan de los recursos las cuales representan los diferentes tipos de recursos (GPIB, Serial...). Cada uno posee atributos y métodos particulares para usar. A continuación se analizan los atributos más comunes.

session: Se trata de un identificador único que representa la comunicación.

timeout: Es el tiempo establecido máximo para una operación y es dado en milisegundos.

chunk_size: Cantidad de fragmentos de datos que lee simultáneamente PyVISA del instrumento. Un fragmento consta de 20 kB.

read_termination: Carácter de terminación de los mensajes.

Escribir y leer valores

Tras establecer la comunicación como previamente se ha explicado, se podrán escribir y leer datos en los instrumentos usando los métodos de la Tabla 6.8.

Tabla 6.8 Métodos empleados para la escritura y lectura en instrumentos.

Método	Descripción
query_ascii_values(data)	Leer valores ASCII
query_binary_values(data, datatype, is_big_endian)	Leer valores binarios
write_ascii_values(data, values)	Escribir valores ASCII
write_binary_values(data, values)	Escribir valores binarios
write(data)	Escribir
read_raw()	Leer

Como se puede observar, los dos últimos métodos de la tabla son generales para leer y escribir. Estos serán los que se emplearán en los códigos de este trabajo. Cuando se quiera escribir algo sobre el instrumento sin esperar respuesta, se empleará el método *write* únicamente, en caso de esperar un dato procedente del instrumento, inmediatamente después se llamará a *read_raw* para almacenar la respuesta. Esto se puede realizar dentro de un bloque *try-except* por si ocurriera algún fallo como se puede ver en el Código 6.32 y en el Código 6.33.

Código 6.32 Ejemplo de lectura de un instrumento.

```
try:
        IV_Obj.write(strCommand)
        Ans = IV_Obj.read_raw()
    except:
        del(IV_Obj)
        print("Failed sending the query.")
```

Código 6.33 Ejemplo de escritura en un instrumento.

```
    try:
        IV_Obj.write(strCommand)
    except:
        del(IV_Obj)
        print("Failed sending the command.")
```

7 Códigos desarrollados

El fracaso es, a veces, más fructífero que el éxito.

Henry Ford

Este capítulo consta del análisis y la explicación del código que se ha redactado y empleado para la toma de medidas en este proyecto. Se trata de una serie de módulos, los cuales cada uno de ellos contienen un listado de definiciones y declaraciones que realizarán una tarea concreta, como por ejemplo, poner un nivel de tensión a una fuente o establecer un límite de intensidad. Los módulos serán analizados en el orden lógico de su escritura, iniciando con lo más básico que es la conexión con el equipo hasta el objetivo final, la toma de datos.

Señalar que muchas de las funciones están duplicadas con la única diferencia del cambio en el número del canal el cual correponderá con el valor 1 si se trata del que da valores negativos de tensión y con el valor 2 si es el que da valores positivos.

En muchas de las funciones se encontrará que se establecerá un valor de 8000 fragmentos al atributo *chunk_size* que tal y como se explicó en el Capítulo 6, determina la cantidad de información a leer simultáneamente proveniente del instrumento. Por otro lado, también será usual realizar una limpieza del registro de estado y de incidencias con el comando **CLS* antes de mandar un comando.

7.1 Apertura de conexión

Antes de enviarle comandos a los equipos para controlarlos remotamente, en el caso de este proyecto más concretamente a la fuente de alimentación HP 6622A, será necesario crear una conexión con dicho equipo. Esto se realizará de la siguiente manera: se llamará a la función *connect_ps* perteneciente al módulo *connect_ps* (Código 7.1) la cual recibe un número de argumentos variables en función de si se quiere modificar el protocolo de comunicación, el número de la tarjeta, la dirección primaria o la dirección secundaria. Si no se le pasan dichos argumentos, se configurará automáticamente con los parámetros por defecto programados, es decir, *GPIB, 0, 5* y sin dirección secundaria respectivamente, lo necesario para controlar la fuente usada en nuestro montaje.

Código 7.1 Inicio de conexión con el instrumento.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 28 21:45:01 2018

@author: Diego
"""

import build_GPIB_object
from six import string_types
```

```
def connect_ps (*varargin):
    numargs = len(locals())
    global Status
    Status = 0
    class init1():
        def __init__ (self, protocol, **GPIB):
            self.protocol = protocol
            self.__dict__.update(GPIB)

    class init2():
        def __init__ (self, board, primaddr):
            self.board = board
            self.primaddr = primaddr
    Parameters = init1 (protocol = 'GPIB', GPIB = init2(0,5))
    if numargs < 1:
        IV_Obj = build_GPIB_object.build_GPIB_object(Parameters)

    if isinstance(varargin[0], string_types) == False:
        raise ("First parameter must be a string.")
    else:
        if varargin[0] == 'GPIB':
            Parameters.protocol = varargin[0]
        else:
            raise ("Protocol type must be GPIB.")

    if numargs > 1:
        if Parameters.protocol == 'GPIB':
            Parameters.GPIB.board = int(varargin[1])

    if numargs > 2:
        Parameters.GPIB.primaddr = int(varargin[2])

    if numargs > 3:
        Parameters.GPIB.secaddr = int(varargin[3])

    IV_Obj = build_GPIB_object.build_GPIB_object(Parameters)

    return Status, IV_Obj
```

Para realizar dicha conexión, será necesaria la creación de un objeto el cual identifique al equipo que se va a controlar. Esto se realiza con la función *build_GPIB_object* del módulo *build_GPIB_object* (Código 7.2) mediante la cual, empleando el módulo *visa*, primero se obtiene un listado de los recursos disponibles mediante el método *ResourceManager* y posteriormente se abre el recurso correspondiente a la fuente, creando el objeto identificador de la misma, el cual ha sido nombrado como *IV_Obj*.

Como comentario adicional, recordar que el atributo *timeout*, explicado en el Capítulo 6, es empleado de manera que si una operación tiene una duración mayor a dicho valor (en milisegundos), ésta es abortada y se lanza una excepción. Se ha establecido dicho atributo a un valor que permita realizar las comunicaciones de manera óptima. Además se ha empleado una bandera que será retornada al realizar la conexión con el equipo mediante la cual se podrá saber si se ha realizado con éxito, tomando el valor 1 en caso afirmativo o el valor 0 en caso contrario.

Código 7.2 Creación del objeto identificador del instrumento.

```
# -*- coding: utf-8 -*-
"""
Created on Sat Apr 28 21:30:04 2018

@author: Diego López Morilla
"""

import visa

def build_GPIB_object(Parameters):
    global Status
    try:
        if hasattr(Parameters.GPIB, 'secaddr'):
            resources = visa.ResourceManager()
            IV_Obj = resources.open_resource(f"{Parameters.protocol}{Parameters.
                GPIB.board}::{Parameters.GPIB.primaddr}::{Parameters.GPIB.
                secaddr}")
            IV_Obj.timeout = 1000
        else:
            resources = visa.ResourceManager()
            IV_Obj = resources.open_resource(f"{Parameters.protocol}{Parameters.
                GPIB.board}::{Parameters.GPIB.primaddr}::INSTR")
            IV_Obj.timeout = 1000
    except:
        raise("Object could not be created")
    Status = 1
    return IV_Obj
```

7.2 Comandos y queries

Como se vio en el Capítulo 5, en la instrumentación virtual se pueden mandar comandos o *queries*. Los primeros se emplean para realizar tareas que no requieran respuesta por parte del equipo como podrían ser establecer un nivel de tensión o de corriente. Por otro lado, las *queries* constan de una comunicación bidireccional, es decir, una vez que el equipo recibe la orden, éste debe responder con algún dato como por ejemplo la identidad del equipo. Para realizar esto, se han creado dos funciones:

send_command

Mediante esta función perteneciente al módulo *send_command* (Código 7.3) se envían comandos a la fuente de alimentación. Recibe como parámetros el objeto que representa al equipo y el comando que se le quiere enviar.

Además, posee ciertas comprobaciones para asegurar que lo que se está enviando se trata de un comando. En caso de fallo en la comunicación a pesar de que el comando sea correcto, se lanzará una excepción.

Código 7.3 Envío de comandos.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 11:21:59 2018

@author: Diego López Morilla
"""

from six import string_types
```

```
def send_command(IV_Obj, strCommand):
    numargs = len(locals())
    Status = 0

    if numargs != 2:
        raise("Number of arguments must be 2.")

    if strCommand == "" or isinstance(strCommand, string_types) == False:
        raise("Command string is empty or not a string.")

    try:
        IV_Obj.write(strCommand)
    except:
        del(IV_Obj)
        print("Failed sending the command.")

    Status = 1
    return Status
```

send_query

Se trata de una función que pertenece al módulo *send_query* (Código 7.4), mediante ella es posible mandar *queries* a la fuente de alimentación. Recibe como parámetros el objeto que representa al equipo y la *query* que se le quiere realizar devolviendo la respuesta a la misma. La estructura es muy similar a la de *send_command*, sin embargo, una vez enviada la cuestión al equipo, se debe leer la respuesta mediante el método *read_raw* visto en el Capítulo 5.

En este caso posee comprobaciones para asegurar que lo que se está enviando se trata de una *query*. En caso de fallo en la comunicación a pesar de que la *query* sea correcta, se lanzará una excepción.

Código 7.4 Envío de *queries*.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 11:47:06 2018

@author: Diego
"""

from six import string_types

def send_query(IV_Obj, strCommand):
    numargs = len(locals())
    Status = 0
    Ans = 0

    if numargs != 2:
        raise("Number of arguments must be 2.")

    if strCommand == "" or isinstance(strCommand, string_types) == False:
        raise("Command string is empty or not a string.")

    if strCommand.find("?") == -1:
        raise("The command is not a query.")

    try:
        IV_Obj.write(strCommand)
```

```
        Ans = IV_Obj.read_raw()
    except:
        del(IV_Obj)
        print("Failed sending the query.")

    Status = 1
    return Status, Ans
```

7.3 Tensión

7.3.1 Manejo de niveles de tensión

Para gestionar los niveles de tensión que va a manejar la fuente, se han escrito una serie de funciones que facilitarán esto. El Código 7.5 se trata del módulo *power_supply_Vpos*, para manejar el canal 2 de la fuente. Por otro lado, el Código 7.6 se corresponde con el módulo *power_supply_Vneg*, destinado al canal 1.

Código 7.5 Manejo de niveles de tensión de la fuente positiva.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 12:02:24 2018

@author: Diego López Morilla
"""
import connect_ps
import send_command
import send_query

def power_supply_Vpos_set(Vin):
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"VSET 2, {Vin}")

    del(ps)

def power_supply_Vpos_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Vout) = send_query.send_query(ps, "VOUT? 2")
    Vout = float(Vout.decode('utf-8'))
    print(f"Voltage (V): {Vout}")

    del(ps)

def power_supply_Vpos_query_r():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Vout) = send_query.send_query(ps, "VOUT? 2")
    Vout = float(Vout.decode('utf-8'))

    del(ps)
    return Vout
```

Código 7.6 Manejo de niveles de tensión de la fuente negativa.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 12:02:24 2018

@author: Diego López Morilla
"""
import connect_ps
import send_command
import send_query

def power_supply_Vneg_set(Vin):
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"VSET 1, {Vin}")

    del(ps)

def power_supply_Vneg_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Vout) = send_query.send_query(ps, "VOUT? 1")
    Vout = float(Vout.decode('utf-8'))
    print(f"Voltage (V): -{Vout}")

    del(ps)

def power_supply_Vneg_query_r():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Vout) = send_query.send_query(ps, "VOUT? 1")
    Vout = float(Vout.decode('utf-8'))

    del(ps)
    return -Vout
```

En dichos módulos se distinguen tres funciones en cada uno de ellos, las cuales tendrán una función concreta y serán explicadas a continuación:

power_supply_Vpos_set / power_supply_Vneg_set
: Mediante esta función se establecerá el nivel de tensión deseado e introducido como argumento en la fuente de alimentación. La unidad del valor tiene que ser el voltio.

power_supply_Vpos_query / power_supply_Vneg_query
: Al ejecutar esta función, se le preguntará a la fuente correspondiente qué valor de tensión está leyendo en ese instante. Dicho valor sera imprimido en la terminal, siendo muy útil a la hora de controlar el equipo ya que se podrá consultar el voltaje que está dando en cada instante con una visualización rápida. La unidad del valor imprimido es el voltio y será indicado junto al dato.

power_supply_Vpos_query_r / power_supply_Vneg_query_r
: Esta función realiza la misma labor que la anterior, sin embargo no imprime el valor de tensión para que sea visualizado por el usuario sino que lo devuelve como parámetro. Esto será de vital importancia en el desarrollo posterior de las pruebas a realizar para poder almacenar los niveles de tensión que están establecidos. La unidad del valor devuelto es el voltio.

7.3.2 Protección de sobretensión

Mediante la protección de sobretensión se podrá fijar una tensión máxima permitida en el equipo, protegiendo el montaje por ejemplo de tensiones indeseadas introducidas accidentalmente o de picos de tensión. El Código 7.7 y el Código 7.8 son los correspondientes a dichas funciones y pertenecen a los módulos *overvoltage_Vpos* y *overvoltage_Vneg* respectivamente.

Código 7.7 Protección de sobretensión de la fuente positiva.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 21:25:17 2018

@author: Diego López Morilla
"""
import connect_ps
import send_command
import send_query

def overvoltage_Vpos_set(Vmax):
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"OVSET 2, {Vmax}")

    del(ps)

def overvoltage_Vpos_rst():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"OVRST 2")

    del(ps)

def overvoltage_Vpos_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Vov) = send_query.send_query(ps, ("OVSET? 2"))
    print(f"Maximum voltage (V): {Vov.decode('utf-8')}" )

    del(ps)
```

Código 7.8 Protección de sobretensión de la fuente negativa.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 21:25:17 2018

@author: Diego López Morilla
"""
import connect_ps
import send_command
import send_query

def overvoltage_Vneg_set(Vmax):
```

```
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"OVSET 1, {Vmax}")

    del(ps)

def overvoltage_Vneg_rst():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"OVRST 1")

    del(ps)

def overvoltage_Vneg_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Vov) = send_query.send_query(ps, ("OVSET? 1"))
    print(f"Maximum voltage (V): {Vov.decode('utf-8')}" )

    del(ps)
```

En dichos módulos se distinguen tres funciones en cada uno de ellos, las cuales tendrán una función concreta y serán explicadas a continuación:

overvoltage_Vpos_set / overvoltage_Vneg_set
: Con esta función se conseguirá inicializar la protección de sobretensión al voltaje deseado pasado como único argumento. La unidad del valor introducido tiene que ser el voltio.

overvoltage_Vpos_rst / overvoltage_Vneg_rst
: Es empleada para reiniciar la protección de sobretensión. Esto será necesario realizarlo si la protección se activa y se llamará tras eliminar la condición de sobretensión.

overvoltage_Vpos_query / overvoltage_Vneg_query
: Esta función imprime el valor máximo de tensión que esté fijado en el instante de su llamada. La unidad del valor devuelto es el voltio.

7.4 Corriente

7.4.1 Manejo de niveles de corriente

Al igual que para el nivel de tensión, también se han escrito un listado de funciones para gestionar la intensidad que proporcionará la fuente. Los módulos empleados para hacerlo serán *power_supply_Ipos* y *power_supply_Ineg* los cuales se corresponden con el Código 7.9 y el Código 7.10 respectivamente.

Código 7.9 Manejo de niveles de corriente de la fuente positiva.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 22:10:04 2018

@author: Diego López Morilla
"""
import connect_ps
import send_command
import send_query
```

```
def power_supply_Ipos_set(Iin):
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"ISET 2, {Iin}")

    del(ps)

def power_supply_Ipos_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Iout) = send_query.send_query(ps, "IOUT? 2")
    Iout = float(Iout.decode('utf-8'))
    Iout = Iout * 1000
    print(f"Current (mA): {Iout}")

    del(ps)

def power_supply_Ipos_query_r():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Iout) = send_query.send_query(ps, "IOUT? 2")
    Iout = float(Iout.decode('utf-8'))
    Iout = Iout * 1000

    del(ps)
    return Iout
```

Código 7.10 Manejo de niveles de corriente de la fuente negativa.

```
Python 3.6.3 |Anaconda, Inc.| (default, Oct 15 2017, 03:27:45) [MSC v.1900 64
    bit (AMD64)] on win32
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 22:10:04 2018

@author: Diego López Morilla
"""
import connect_ps
import send_command
import send_query

def power_supply_Ineg_set(Iin):
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"ISET 1, {Iin}")

    del(ps)

def power_supply_Ineg_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Iout) = send_query.send_query(ps, "IOUT? 1")
    Iout = float(Iout.decode('utf-8'))
```

```
    Iout = Iout * 1000
    print(f"Current (mA): {Iout}")

    del(ps)

def power_supply_Ineg_query_r():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, Iout) = send_query.send_query(ps, "IOUT? 1")
    Iout = float(Iout.decode('utf-8'))
    Iout = Iout * 1000

    del(ps)
    return Iout
```

En dichos módulos se distinguen tres funciones en cada uno de ellos, las cuales tendrán una función concreta y serán explicadas a continuación:

power_supply_Ipos_set / power_supply_Ineg_set
: Mediante esta función se introducirá como único argumento la intensidad máxima deseada que puede dar la fuente para que no dañe los dispositivos empleados. La unidad del valor introducido tiene que ser el miliamperio.

power_supply_Ipos_query / power_supply_Ineg_query
: Cuando se ejecute, preguntará a la fuente qué valor de intensidad está leyendo. La respuesta a la pregunta será imprimida por pantalla para que el usuario pueda visualizarla. La unidad del valor imprimido es el miliamperio y será indicado junto al dato.

power_supply_Ipos_query_r / power_supply_Ineg_query_r
: Al igual que ocurre en tensión, en corriente existe una función que pregunta el valor pero no lo imprime por pantalla, sino que lo devuelve para almacenarlo en una variable. Esto será de vital importancia en el desarrollo posterior de las pruebas a realizar para poder almacenar los niveles de corriente que están establecidos. La unidad del valor devuelto es el miliamperio.

7.4.2 Protección de sobrecorriente

La protección de sobrecorriente controla que no pase más intensidad que la deseada para que no se quemen los dispositivos que componen el montaje. En el Código 7.11 se pueden observar las funciones pertenecientes al módulo *overcurrent_Ipos* y en el Código 7.12 los del *overcurrent_Ineg*.

Código 7.11 Protección de sobrecorriente de la fuente positiva.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 21:33:07 2018

@author: Diego López Morilla
"""

import connect_ps
import send_command
import send_query

def overcurrent_Ipos_set(OCP):
    if OCP == 'ON':
        state = 1
    else:
        if OCP == 'OFF':
            state = 0
```

```
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"OCP 2, {state}")

    del(ps)

def overcurrent_Ipos_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, OCPstate) = send_query.send_query(ps, "OCP? 2")
    print(f"Overcurrent protection state: {OCPstate.decode('utf-8')}")

    del(ps)
```

Código 7.12 Protección de sobrecorriente de la fuente negativa.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Apr 29 21:33:07 2018

@author: Diego
"""
import connect_ps
import send_command
import send_query

def overcurrent(OCP):
    if OCP == 'ON':
        state = 1
    else:
        if OCP == 'OFF':
            state = 0

    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    send_command.send_command(ps, ("*CLS"))
    send_command.send_command(ps, f"OCP 1, {state}")

    del(ps)

def overcurrent_Ineg_query():
    (Status, ps) = connect_ps.connect_ps('GPIB')
    ps.chunksize = 8000
    (Status, OCPstate) = send_query.send_query(ps, "OCP? 1")
    print(f"Overcurrent protection state: {OCPstate.decode('utf-8')}")

    del(ps)
```

7.4.3 Códigos orientados a las pruebas

Para realizar las pruebas se han elaborado tres módulos que ayudarán al desarrollo de las mismas. Se trata de *power_supply_Imax*, *power_supply_measure* y *power_supply*.

El primero contiene una única función que recibe como primer argumento la intensidad máxima en miliamperios que se desea que proporcione el canal 1 y como segundo lo análogo respecto al canal 2 de la fuente. Dicho método puede consultarse en el Código 7.13.

Código 7.13 Limitación de corriente en ambos canales de la fuente.

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 30 00:09:04 2018

@author: Diego López Morilla
"""
import power_supply_Ineg
import power_supply_Ipos

def power_supply_Imax_set(Imax_neg, Imax_pos):
    power_supply_Ineg.power_supply_Ineg_set(Imax_neg/1000)
    power_supply_Ipos.power_supply_Ipos_set(Imax_pos/1000)
```

El siguiente, *power_supply_measure*, contiene también una única función la cual al llamarla proporcionará al usuario por pantalla una información muy completa acerca de los valores de tensión e intensidad que está proporcionando la fuente en todos sus canales. Esto se realizará con el Código 7.14.

Código 7.14 Medida de tensión y corriente en ambos canales de la fuente.

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 31 23:30:47 2018

@author: Diego López Morilla
"""
import power_supply_Vpos
import power_supply_Vneg
import power_supply_Ipos
import power_supply_Ineg

def power_supply_measure():

    print("Power supply 1 (Negative):")
    power_supply_Vneg.power_supply_Vneg_query();
    power_supply_Ineg.power_supply_Ineg_query();

    print("Power supply 2 (Positive):")
    power_supply_Vpos.power_supply_Vpos_query();
    power_supply_Ipos.power_supply_Ipos_query();
```

El último se muestra en el Código 7.15 y está compuesto de una única función con el mismo nombre que el módulo la cual recibirá tres parámetros en el siguiente orden: número de canales de la fuente que se quieren emplear (en el caso de este proyecto deberá ser 1 ó 2 ya que posee dos canales), tensión del canal 2 de la fuente (será la tensión V_{ds}) y tensión del canal 1 de la fuente (será la V_{gs}).

Código 7.15 Fijado de tensión de los canales de la fuente.

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 30 00:17:22 2018

@author: Diego López Morilla
"""
import power_supply_Vpos
import power_supply_Vneg
import power_supply_measure

def power_supply(nps, Vds, Vgs):
    numargs = len(locals())
    if numargs != 3:
        print("The number of arguments must be 3.")
    else:
        if nps == 1:
            if not Vgs:
                while Vds < 0:
                    print("ATTENTION: Vds is negative.")
                    Vds = input("Please, enter a positive value (V).")
                power_supply_Vpos.power_supply_Vpos_set(Vds)
            elif not Vds:
                while Vgs > 0:
                    print("ATTENTION: Vgs is positive.")
                    Vgs = input("Please, enter a negative value (V).")
                power_supply_Vneg.power_supply_Vneg_set(-Vgs)
            else:
                print("Wrong voltage.")
                print("Please, enter new values.")
        elif nps == 2:
            while Vds < 0:
                print("ATTENTION: Vds is negative.")
                Vds = input("Please, enter a positive value (V).")
            while Vgs > 0:
                print("ATTENTION: Vgs is positive.")
                Vgs = input("Please, enter a negative value (V).")
            power_supply_Vpos.power_supply_Vpos_set(Vds)
            power_supply_Vneg.power_supply_Vneg_set(-Vgs)
        else:
            print("The number of power supplies must be 1 or 2.")

        power_supply_measure.power_supply_measure()
```

8 Pruebas experimentales

La creatividad requiere el coraje de dejar ir las certezas.

Erich Fromm

Tras analizar en el Capítulo 7 los códigos básicos escritos para el desarrollo del proyecto, en el presente se explicarán de qué manera se combinarán todos ellos para realizar una serie de pruebas prácticas en el laboratorio.

La primera prueba consistirá en poner a trabajar un transistor de Cree en su punto óptimo de polarización. Se proseguirá con una prueba cuyo objetivo es realizar un barrido de tensiones para caracterizar el comportamiento I-V de un transistor FET. Por último, se realizará un script para controlar en tiempo real el valor que está tomando la tensión e intensidad proporcionada por la fuente en ambos canales. Antes de realizar cualquiera de estas pruebas será necesario importar todos los módulos creados para poder llamarlos desde la terminal. Se ha creado un módulo llamado *import* el cual tiene como única funcionalidad la importación del resto de módulos:

Código 8.1 Importación de los módulos.

```
# -*- coding: utf-8 -*-
"""
Created on Wed May 16 13:07:16 2018

@author: Diego López Morilla
"""

import power_supply_Vpos
import power_supply_Vneg
import power_supply_Ipos
import power_supply_Ineg
import overvoltage_Vpos
import overvoltage_Vneg
import overcurrent_Ipos
import overcurrent_Ineg

import send_query
import send_command
import build_GPIB_object
import connect_ps
import power_supply_Imax
import power_supply_measure
import power_supply
import graphics_TR
```

8.1 Prueba 1: Búsqueda del punto de polarización de un amplificador

En esta prueba se empleará una placa de evaluación de un amplificador de Cree basado en el transistor HEMT (*high electron mobility transistor*) CGH40010. En la Figura 8.1 se puede observar dicha placa y en la Figura 8.2 el transistor individualmente fuera de la misma.

El transistor CGH40010 es un transistor de nitruro de galio (GaN) de gran movilidad de electrones. El CGH40010 ofrece una solución de uso general para una variedad de aplicaciones de RF y microondas dando una alta eficiencia, alta ganancia y gran ancho de banda, lo que hace que el CGH40010 sea ideal para circuitos amplificadores lineales y comprimidos. La placa empleada está optimizada para trabajar a una frecuencia de 3.6 GHz, dando una ganacia de 13 dB. El punto recomendado para polarizar el transistor es el siguiente:

- V_{ds} = 28 V.
- I_{ds} = 200 mA.
- V_{gs} = -2.36 V.

El procedimiento recomendado para alimentar transistores HEMT o FET de empobrecimiento, es decir, con un valor de V_{gs} negativo, es el siguiente:

1. Fijar la intensidad máxima de los canales.
2. Inicializar las tensiones V_{ds} y V_{gs} a valores seguro, esto es, con una tensión V_{gs} negativa y por debajo del valor de corte.
3. Realizar el conexionado del montaje.
4. Aumentar las tensiones progresivamente hasta llegar al punto óptimo de polarización.

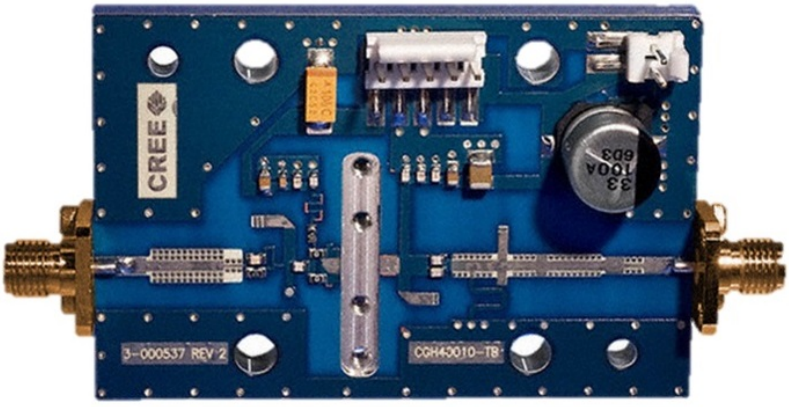


Figura 8.1 Placa de evaluación del amplificador de Cree.

Figura 8.2 HEMT CGH40010.

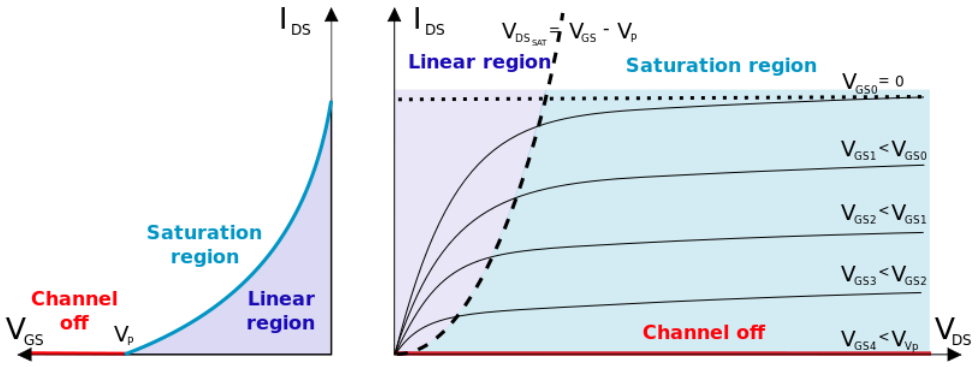
Esta prueba ha consistido en realizar los pasos que previamente se han expuesto para llegar al punto de polarización recomendado por el fabricante. El montaje necesario para llevarla a cabo se muestra en la Figura 8.3.

Figura 8.3 Montaje para la primera prueba en el laboratorio.

En la Figura 8.5 se muestran las interacciones a realizar en la terminal para realizar la prueba. Primeramente se fijan 5 mA y 400 mA como intensidades I_{gs} e I_{ds} máximas permitidas para asegurar que no se queme el DUT. Posteriormente se fijarán las tensiones V_{ds} y V_{gs} a 0 V y -4 V respectivamente para realizar el conexionado del montaje sin riesgo. Por último se irán aumentando ambas tensiones progresivamente hasta alcanzar los 28 V de V_{ds} y -2.36 V de V_{gs} como se mencionó previamente.

Paralelamente se irán realizando medidas para comprobar que las intensidades toman valores correctos y no se han disparado a un valor muy elevado, lo que sería síntoma de un incorrecto funcionamiento del dispositivo bajo prueba. Más concretamente, mientras la V_{gs} está por debajo de la tensión de corte, que según el *datasheet* (Apéndice B) está en el rango de -3.8 a -2.3 V con un valor típico de -3.3 V, se puede apreciar que el transistor está al corte y las corrientes I_{ds} e I_{gs} son prácticamente nulas. Por el contrario, cuando aumenta V_{gs} y supera ese umbral, el transistor empieza a conducir, obteniéndose mayor I_{ds} al aumentar V_{gs} para un valor de V_{ds} fijo, como es el caso. Finalmente, cuando el valor de V_{gs} vuelve a estar por debajo de la tensión de corte, la corriente I_{ds} de nuevo es prácticamente nula porque se ha vuelto a cortar el canal.

Éste es el comportamiento esperado del transistor tal y como se representa en la Figura 8.4, donde se puede consultar una gráfica del comportamiento teórico que debe tener un transistor JFET en sus tres estados de funcionamiento: corte, lineal y saturación.



(a) Curva I_{ds}-V_{gs} teórica de un transistor JFET.

(b) Curva I_{ds}-V_{ds} teórica de un transistor JFET.

Figura 8.4 Comportamiento teórico de un transistor JFET.

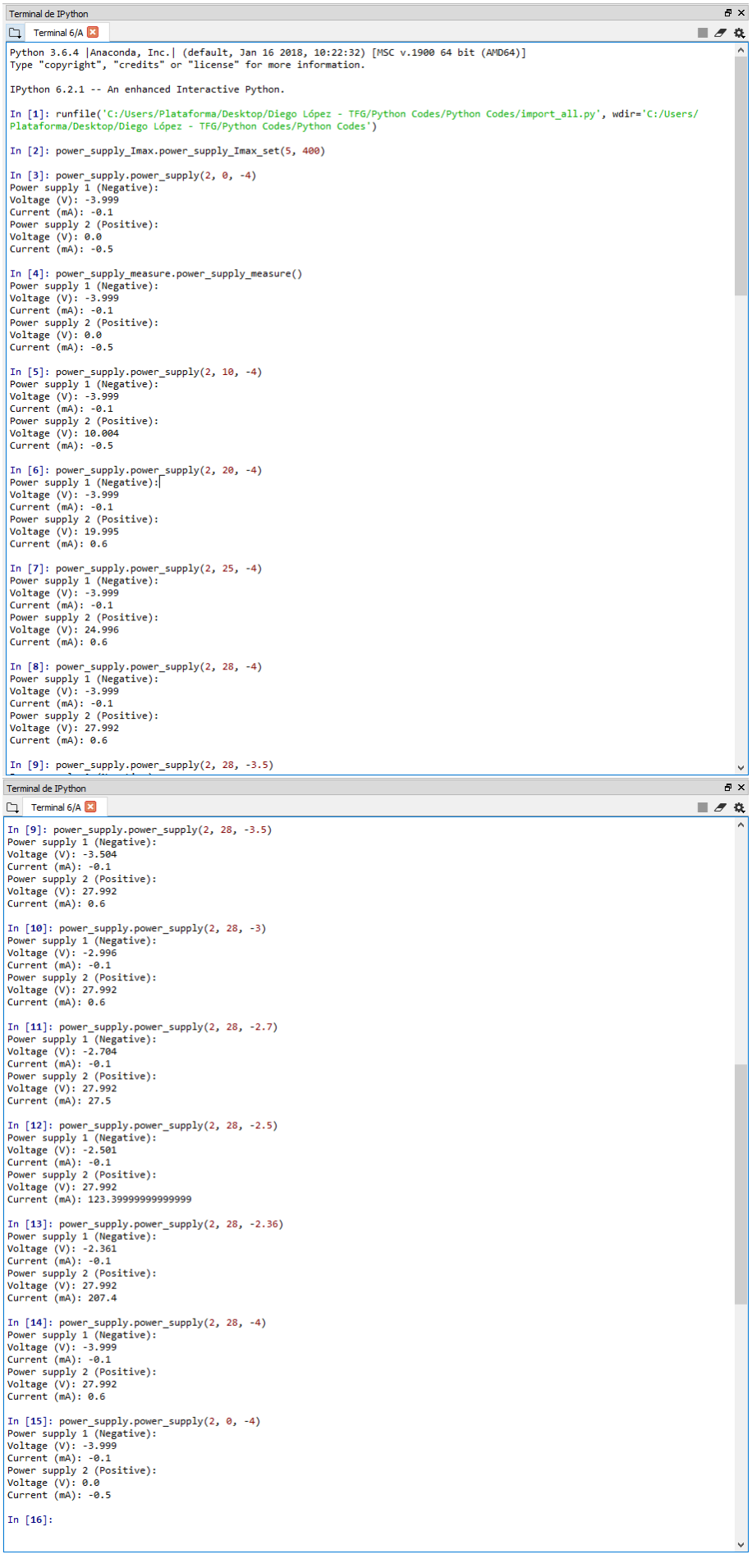


Figura 8.5 Interacciones para el desarrollo de la primera prueba experimental.

8.2 Prueba 2: Barrido de tensión

Esta prueba consiste en realizar un doble barrido de tensión aplicado a un transistor FET. Se ha escrito un script llamado *graphic_IV* que, al ejecutarlo, hará al usuario introducir los valores mínimos y máximos de tensión (tanto V_{ds} como V_{gs}) y el salto de una tensión a otra, creando dos vectores de tensiones para realizar el barrido posterior. Esto se realiza con el Código 8.2.

Código 8.2 Doble barrido de tensión.

```
# -*- coding: utf-8 -*-
"""
Created on Thu May 31 23:40:40 2018

@author: Diego López Morilla
"""

import time
import matplotlib.pyplot as plt  # Cargamos matplotlib.pyplot como plt
import power_supply_Vpos
import power_supply_Vneg
import power_supply_Ipos
import power_supply_Ineg
import power_supply
import numpy as np   # Cargamos numpy como np
from random import randrange

def comp_number(n):
    try:
        float(n)
        return False
    except:
        print("You must enter a number.")
        return True

print("Units: Volts")

Vds_min = input ("Minimum value (Vds): ")
while comp_number(Vds_min) or float(Vds_min) < 0:
    Vds_min = input ("Minimum value (Vds): ")

Vds_step = input ("Step value (Vds): ")
while comp_number(Vds_step) or float(Vds_min) < 0:
    Vds_step = input ("Step value (Vds): ")

Vds_max = input ("Maximum value (Vds): ")
while comp_number(Vds_max) or float(Vds_min) < 0:
    Vds_max = input ("Maximum value (Vds): ")

Vgs_min = input ("Minimum value (Vgs): ")
while comp_number(Vgs_min) or float(Vgs_min) > 0:
    Vgs_min = input ("Minimum value (Vgs): ")

Vgs_step = input ("Step value (Vgs): ")
while comp_number(Vgs_step) or float(Vgs_step) < 0:
    Vgs_step = input ("Step value (Vgs): ")
```

```
Vgs_max = input ("Maximum value (Vgs): ")
while comp_number(Vgs_max) or float(Vgs_max) > 0:
    Vgs_max = input ("Maximum value (Vgs): ")

Vds_in = np.arange(float(Vds_min), float(Vds_max) + 0.00000001, float(Vds_step)
    )
Vgs_in = np.arange(float(Vgs_min), float(Vgs_max) + 0.00000001, float(Vgs_step)
    )
Ids_out = [0] * len(Vds_in)
Igs_out = [0] * len(Vds_in)
Vds_out = [0] * len(Vds_in)
Vgs_out = [0] * len(Vds_in)
colours = ["b", "g", "k", "r", "c", "m", "y"]
legend = []
ii=jj=0
filename = input("File name:")
file = open(f"{filename}", "w")
plt.figure()
fig, ((ax1, ax2)) = plt.subplots(nrows=2, ncols=1, figsize=(10, 8))
for i in Vgs_in:
    jj=0
    for j in Vds_in:
        power_supply.power_supply(2, j, i)
        time.sleep(0.1)
        Vgs_out[jj] = power_supply_Vneg.power_supply_Vneg_query_r()
        Vds_out[jj] = power_supply_Vpos.power_supply_Vpos_query_r()
        Igs_out[jj] = power_supply_Ineg.power_supply_Ineg_query_r()
        Ids_out[jj] = power_supply_Ipos.power_supply_Ipos_query_r()
        jj = jj + 1
    print("Vgs = %.2f" %i)  # Impresión del valor de tensión Vgs para comprobar
         funcionamiento
    # Se exportan los valores al archivo externo
    file.write(f"Vgs = {i}\n")
    file.write(f"Vgs_out = {Vgs_out}\n")
    file.write(f"Vds_out = {Vds_out}\n")
    file.write(f"Igs_out = {Igs_out}\n")
    file.write(f"Ids_out = {Ids_out}\n")

    ax1.plot(Vds_out, Ids_out, colours[randrange(6)], linewidth = 2)
    index = str(i)
    legend.append("Vgs = %.2f" %i)
    ax2.plot(Vds_out, Igs_out, colours[randrange(6)], linewidth = 2)
    plt.hold(True)
    ii = ii + 1

file.close()

ax1.locator_params(nbins=3)
ax1.set_xlabel("Vds", fontsize=12)
ax1.set_ylabel("Ids", fontsize=12)

box1 = ax1.get_position()
ax1.set_position([box1.x0, box1.y0, box1.width * 0.8, box1.height])

ax1.legend(legend, loc='center left', bbox_to_anchor=(1, 0.5))
ax2.locator_params(nbins=3)
ax2.set_xlabel("Vds", fontsize=12)
```

```
ax2.set_ylabel("Igs", fontsize=12)

box2 = ax2.get_position()
ax2.set_position([box2.x0, box2.y0, box2.width * 0.8, box2.height])

ax2.legend(legend, loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

El montaje para dicha prueba se puede observar en la Figura 8.6, en el que se ha empleado el transistor EPB018A5-70, el cual se trata de un transistor FET de heterounión que proporciona una gran ganancia dando un muy bajo ruido.

Para que no existan problemas con el transistor que se va a emplear, se ha consultado su *datasheet* (Apéndice C) observando unos valores máximos que habrá que tener en cuenta a la hora de realizar pruebas con él:

- $V_{ds_{max}}$: 4 V.
- $V_{gs_{max}}$: -2 V.
- $I_{ds_{max}}$: 60 mA.
- $I_{gs_{max}}$: 0.3 mA.
- P_{diss}: 240 mW.

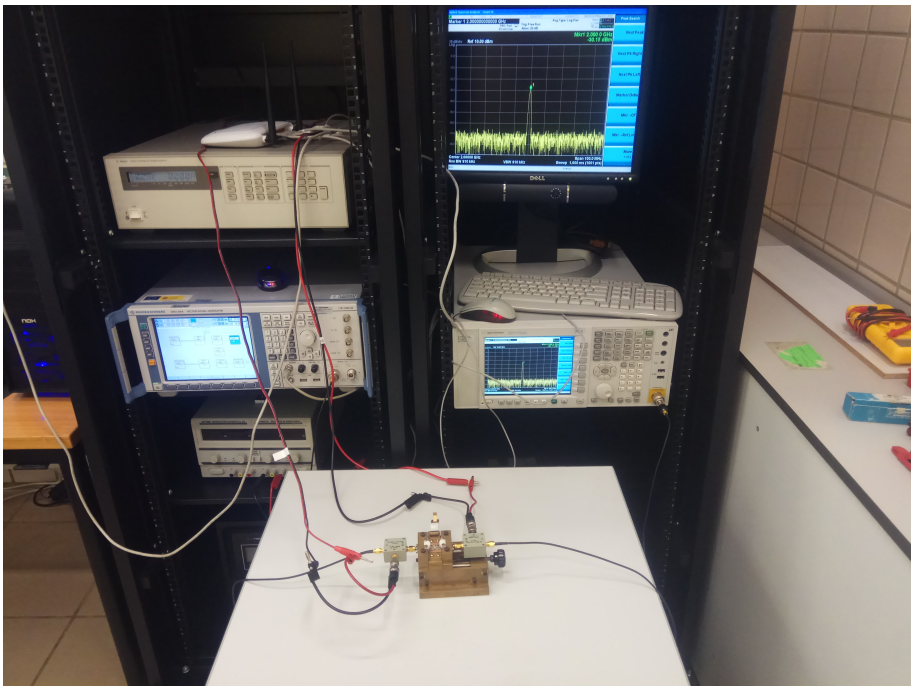


Figura 8.6 Montaje para la segunda prueba en el laboratorio.

Como se ha explicado antes, se fijarán las tensiones para realizar el doble barrido. El barrido consiste, más concretamente, en fijar un valor de V_{gs} y, con ese valor fijado, barrer todos los valores posibles de V_{ds} leyendo con cada pareja de valores de tensiones establecidos las intensidades correspondientes. Tras esto se imprimirán las gráficas V_{ds} frente a I_{ds} y V_{ds} frente a I_{gs} y se cambiará el valor de tensión de V_{gs}, empezando de nuevo el ciclo. Con esto se conseguirá caracterizar el transistor mediante sus gráficas I-V.

Para el caso particular de este proyecto el barrido realizado es empleando el rango de tensiones mostrado en la Tabla 8.1. La introducción de dichos valores en el script se ilustra en la Figura 8.7. Una vez hecho esto, se introducirá un nombre para el archivo en el que se irán volcando los valores de tensiones e intensidades para poder exportarlos y gestionarlos posteriormente.

Las gráficas del comportamiento I-V, que son el resultado de esta prueba experimental, se pueden ver en la Figura 8.8, que son el objetivo de esta prueba. Los valores obtenidos en este experimento, los cuales son empleados para realizar la representación gráfica se pueden consultar en el Apéndice A.

Tabla 8.1 Valores del barrido.

	Mínimo	Máximo	Paso
V_{ds}	0	3.5	0.05
V_{gs}	-1	0	0.1

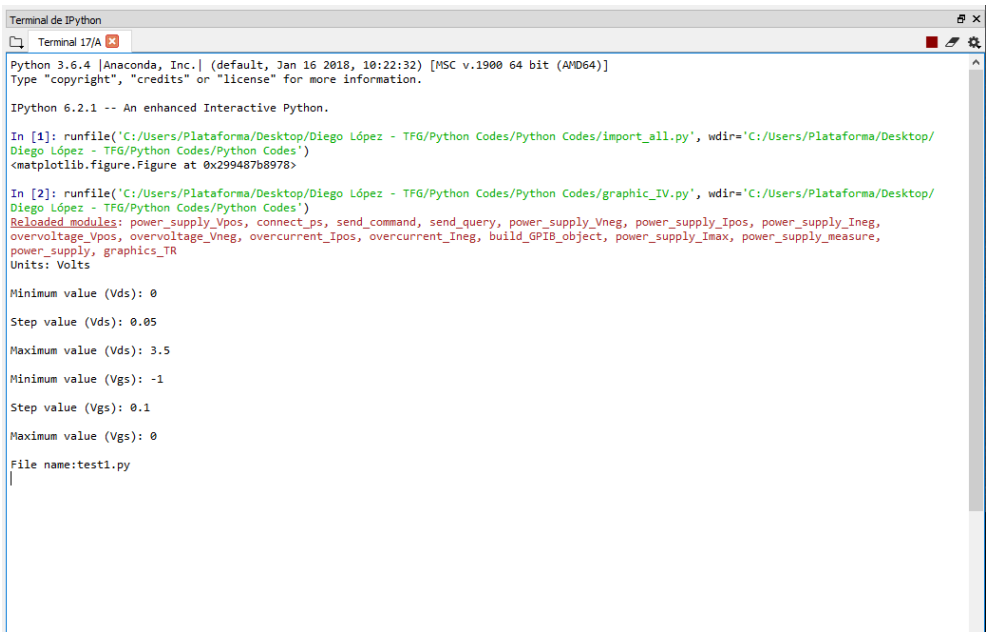


Figura 8.7 Interacciones para el desarrollo de la segunda prueba experimental.

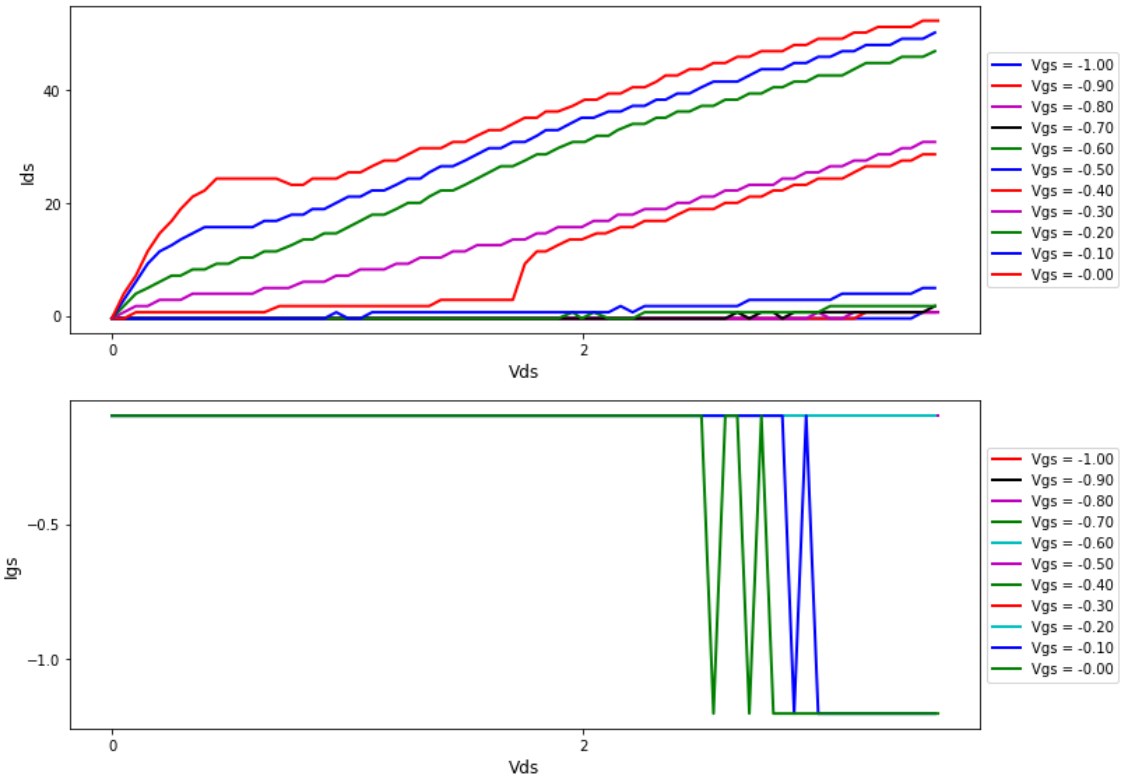


Figura 8.8 Característica I-V del transistor.

El resultado obtenido es coherente con lo esperado ya que sigue el comportamiento teórico lógico. Haciendo referencia a la Figura 8.4b se puede apreciar la similitud con la gráfica generada en esta prueba, diferenciándose las distintas zonas de funcionamiento del transistor. Primeramente, los valores menores de tensión V_{gs} son insuficientes para hacer activar el transistor, por lo que está cortado y se corresponde con un valor de corriente I_{ds} casi nulo. Posteriormente y a partir de una tensión de puerta concreta, podrá conducir en lineal o saturación según el valor de V_{ds} asignado en cada iteración. Por otro lado, también se puede observar cómo se cumple lo representado en la Figura 8.4a, ya que al aumentar la tensión V_{gs} se puede apreciar cómo la intensidad I_{ds} también se incrementa.

Por último, hacer referencia a la gráfica correspondiente con el valor de V_{gs} = -0.4 V, ya que presenta un salto bastante anómalo en lugar de seguir la tendencia del resto de gráficas. Otra peculiaridad que se puede observar de las medidas tomadas está en la gráfica de V_{gs} = 0 V, donde se observa una subida y después bajada de la intensidad I_{ds} en torno al valor de V_{ds} = 0.5 V. Además, la precisión con la que se mide la corriente en la fuente de alimentación no es la óptima, y eso se ve reflejado en un pequeño rizado en el valor de la I_{ds} en todas las medidas.

8.3 Prueba 3: Monitoreo en tiempo real

Tal y como se ha introducido previamente, esta prueba consistirá en la elaboración de un código que esté mostrando gráficas en tiempo real de los valores que están tomando las tensiones (V_{gs} y V_{ds}) e intensidades (I_{gs} e I_{ds}) en todo momento.

La finalidad principal de este experimento es poder observar en tiempo real los valores que está dando la fuente y así, si se observa alguna anomalía en las gráficas, el usuario sabrá que algo ha ocurrido. Para llevar a cabo esta tarea se ha desarrollado el Código 8.3 el cual contiene una única función que recibe como argumentos los valores máximos y mínimos de cada eje vertical.

Código 8.3 Monitoreo en tiempo real.

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 1 00:42:47 2018

@author: Diego López Morilla
"""

import time
import matplotlib.pyplot as plt  # Cargamos matplotlib.pyplot como plt
import power_supply_Vpos
import power_supply_Vneg
import power_supply_Ipos
import power_supply_Ineg
x = 500
i=1
tms = list(range(0,x,1))
Vds = [0] * x
Vgs = [0] * x
Ids = [0] * x
Igs = [0] * x
init = time.time()
plt.figure()
def graphics_TR(ymin_Vgs, ymax_Vgs, ymin_Igs, ymax_Igs, ymin_Vds, ymax_Vds,
    ymin_Ids, ymax_Ids):
    while 1:
        time.sleep(0.1)
        #plt.close()
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(nrows=2, ncols=2)

        ax1.plot(tms, Vgs)
        ax1.locator_params(nbins=3)
        ax1.set_xlabel("t", fontsize=12)
        ax1.set_ylabel("Vgs", fontsize=12)
        ax1.axis((min(tms), max(tms), ymin_Vgs, ymax_Vgs))

        ax2.plot(tms, Igs)
        ax2.locator_params(nbins=3)
        ax2.set_xlabel("t", fontsize=12)
        ax2.set_ylabel("Igs", fontsize=12)
        ax2.axis((min(tms), max(tms), ymin_Igs, ymax_Igs))

        ax3.plot(tms, Vds)
        ax3.locator_params(nbins=3)
        ax3.set_xlabel("t", fontsize=12)
        ax3.set_ylabel("Vds", fontsize=12)
        ax3.axis((min(tms), max(tms), ymin_Vds, ymax_Vds))

        ax4.plot(tms, Ids)
        ax4.locator_params(nbins=3)
        ax4.set_xlabel("t", fontsize=12)
        ax4.set_ylabel("Ids", fontsize=12)
        ax4.axis((min(tms), max(tms), ymin_Ids, ymax_Ids))

        plt.tight_layout()
```

```
        plt.show()
        tms.append(x + (time.time()-init))
        tms.remove(tms[0])


        Vds.append(power_supply_Vpos.power_supply_Vpos_query_r())
        Vds.remove(Vds[0])
        Ids.append(power_supply_Ipos.power_supply_Ipos_query_r())
        Ids.remove(Ids[0])
        Vgs.append(power_supply_Vneg.power_supply_Vneg_query_r())
        Vgs.remove(Vgs[0])
        Igs.append(power_supply_Ineg.power_supply_Ineg_query_r())
        Igs.remove(Igs[0])
```

El script básicamente preguntará por los valores de tensión e intensidad que estén establecidos en cada momento en los canales e irá dibujando gráficas una tras otra eliminando los valores antiguos e introduciendo los más actuales.

El dispositivo que se va a emplear en esta prueba es el amplificador de baja potencia MAX2430 de Maxim cuya banda de frecuencia de funcionamiento es entre 800 MHz y 1000 MHz. Dicho amplificador se puede alimentar con una tensión entre 3 V y 5.5 V. En este caso se le aplicará una tensión de alimentación de 3.6 V. La eficiencia del amplificador es del 24 % y proporciona ganancias por encima de los 32 dB.

La particularidad que tiene este dispositivo es que se alimenta con una tensión positiva constante pero la corriente que consume cambia al variar la potencia de RF a la entrada. Por ejemplo, con un valor de Vcc=5 V y sin entrada de RF, el amplificador estaría consumiendo 52 mA aproximadamente, valor que aumentaría una vez se le introduzca la RF. El montaje empleado se puede ver en la Figura 8.9 visualizándose individualmente el amplificador en la Figura 8.10.

Figura 8.9 Montaje para la tercera prueba en el laboratorio.

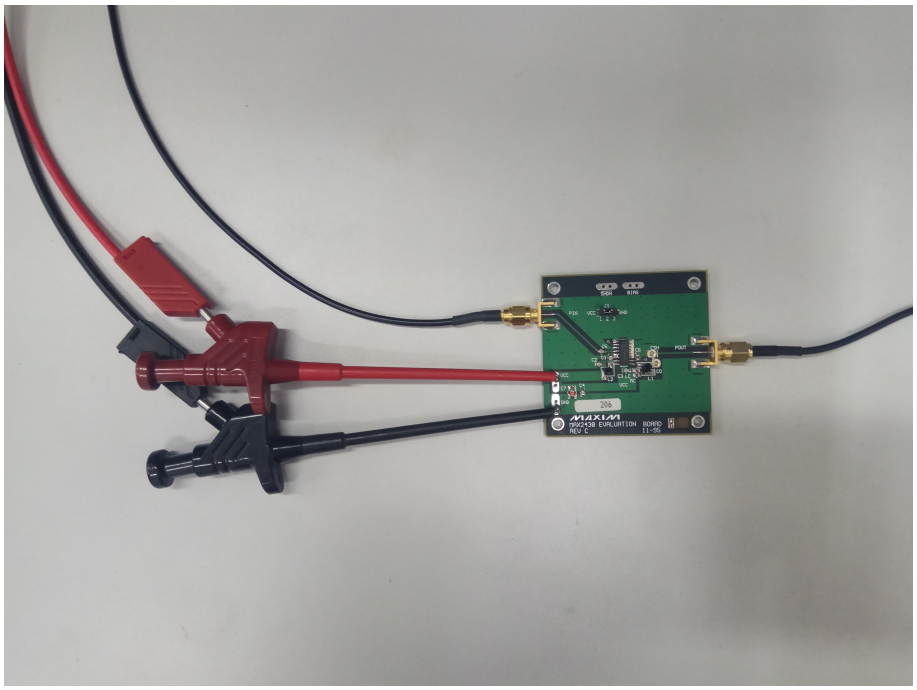


Figura 8.10 Amplificador MAX2430.

El objetivo de esta prueba es poder visualizar en las gráficas esa variación de consumo que realiza el dispositivo, para ello se han empleado dos terminales, una en la que se ejecute el script que imprime las gráficas continuamente (Figura 8.12), donde se han usado como límites de los ejes los mostrados en la Tabla 8.2 y en la otra en la que se introducirán los datos (Figura 8.13). Para conseguir el objetivo deseado, se aplicarán distintos valores de nivel de RF a la entrada al dispositivo para que vaya cambiando la intensidad que consume.

Antes de empezar, se deberá configurar que las gráficas se impriman en una ventana nueva en lugar de en la terminal para poder visualizarlas de una manera fluida. Haciendo click en *Herramientas-Preferencias* se abrirá una ventana nueva, en la que habrá que seleccionar *Terminal de IPyhon* y en la pestaña *Gráficas* se seleccionará *Salida: Automático* tal y como se observa en la Figura 8.11.

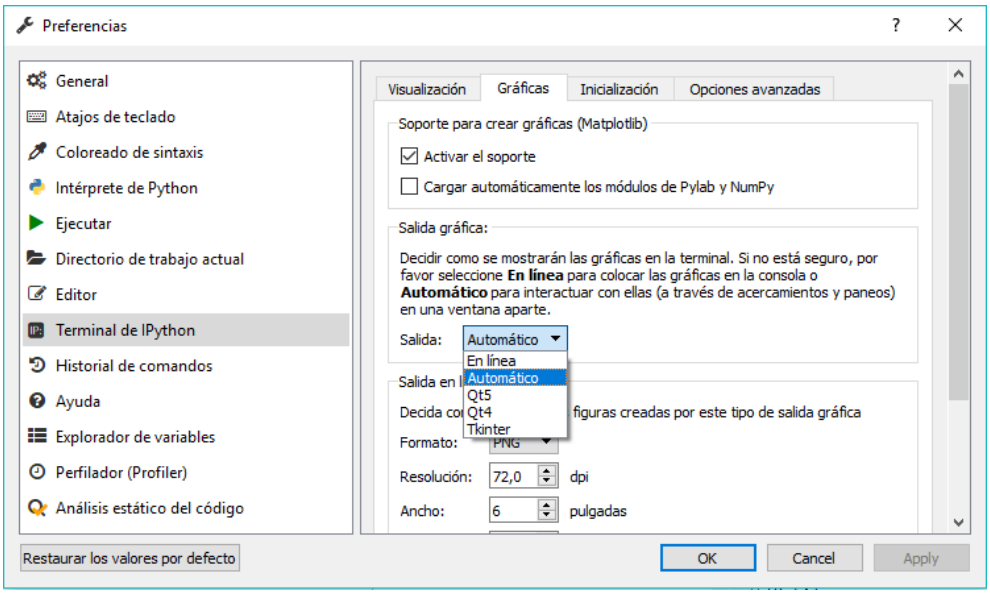


Figura 8.11 Configuración de Spyder para las gráficas.

Inicialmente se limita en intensidad a un valor de 300 mA para evitar dañar el dispositivo. Posteriormente se inicia el valor de tensión a 0 V, por lo que el consumo será de 0 A tal y como se puede observar en la gráfica de la Figura 8.14.

A continuación se alimentará el amplificador y con el generador de señal se irá variando manualmente el nivel de RF, ya que no tenemos control remoto del generador de señal. De esta manera se pueden observar cambios en el nivel de intensidad.

Por último, se deja de alimentar al dispositivo estableciendo una tensión de 0 V, por lo que dejará de consumir. Cabe destacar que los valores de V_{gs} e I_{gs} son nulos todo el tiempo porque no se está empleando el canal correspondiente de la fuente en esta prueba.

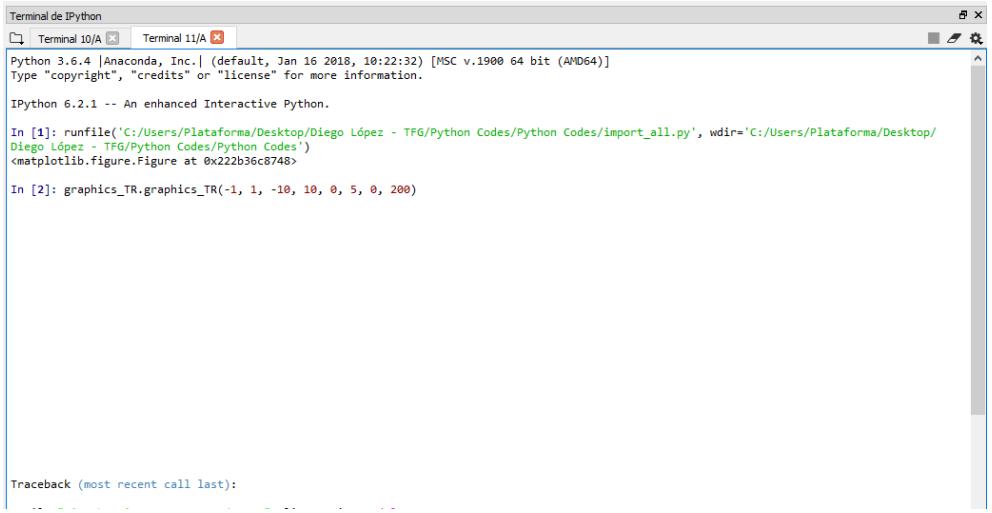


Figura 8.12 Llamada al script para la apertura de las gráficas.

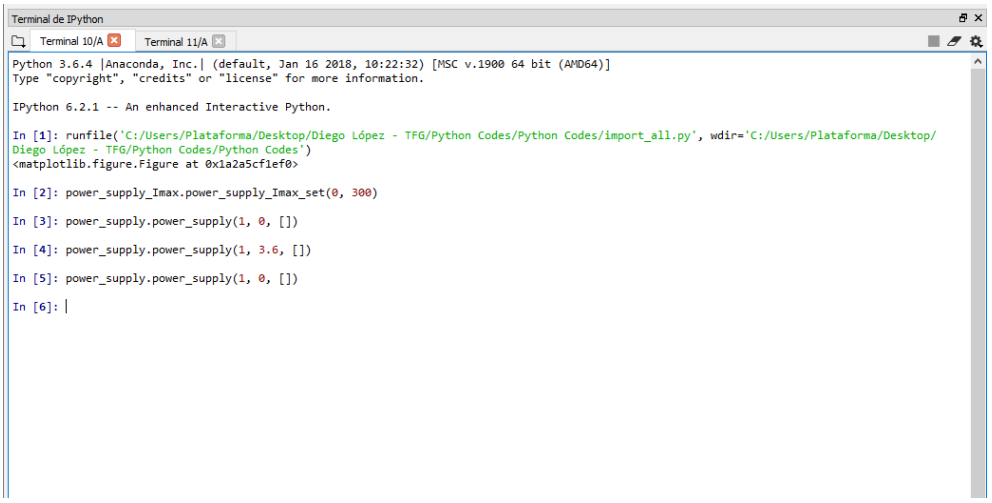


Figura 8.13 Interacciones para el desarrollo de la tercera prueba experimental.

Tabla 8.2 Tamaño de los ejes.

	Mínimo	**Máximo**
V_{gs}	-1	1
I_{gs}	-10	10
V_{ds}	0	5
I_{ds}	0	200

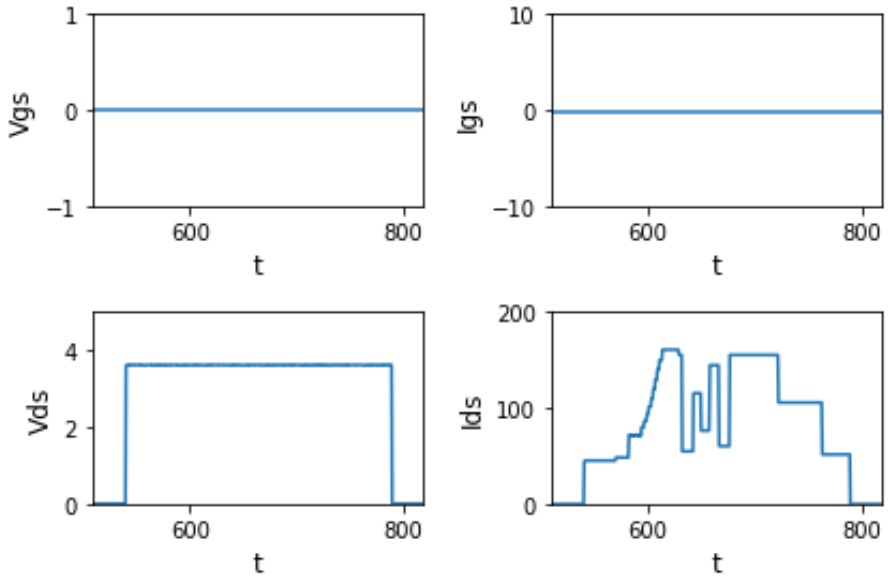


Figura 8.14 Gráfica para el análisis del consumo del MAX2430.

9 Conclusiones y líneas futuras de trabajo

Pero hay que seguir adelante. Las batallas hay que darlas se ganen o se pierdan; hay que darlas por el hecho mismo de darlas. Porque eso nos cumple, eso nos ratifica.

José Luis Sampedro

En este proyecto se ha dado especial énfasis al concepto de instrumentación virtual, el cual cada vez se encuentra más presente en los proyectos de toma de datos y actualmente se han desarrollado muchísimos sistemas para implementar aplicaciones que se basan en instrumentación virtual. Esto es debido a los numerosos beneficios y a la eficiencia que proporciona esta metodología de trabajo, permitiendo al usuario configurar y generar sus propios sistemas consiguiendo un buen desempeño de los equipos acompañado de una alta flexibilidad, reutilización y reconfiguración de los mismos, de manera que se reducen considerablemente los costes de desarrollo, mantenimiento, etc.

Este concepto asegura, además de que el trabajo actual podrá ser empleado en el futuro, una flexibilidad y extensión a medida que cambian las necesidades funcionales y operativas del usuario, ya que al estar todo definido por software y no por hardware, la adaptación del instrumento según la necesidad es casi completa.

Además se han explicado los diferentes buses existentes de comunicación utilizados en la instrumentación virtual que se pueden encontrar en la actualidad, haciendo un breve resumen de ellos como por ejemplo de RS 232, PXI... En este trabajo se ha empleado el bus GPIB, uno de los más usados en la actualidad debido a las altas prestaciones que ofrece. Para la comunicación con los equipos se ha hecho uso de comandos SCPI con el apoyo del módulo PyVISA de Python para enviar dichos comandos. Python ha proporcionado ahorro de tiempo y facilidad de programación, al tratarse de un lenguaje interpretado.

En este proyecto se ha desarrollado un conjunto de códigos muy potentes mediante los cuales se consigue reducir notablemente el tiempo del usuario en la realización de medidas, ya que si no se contara con ellos, se hubieran tenido que introducir manualmente los valores correspondientes en cada medida.

Los códigos desarrollados en este proyecto pueden ser empleados para la realización de múltiples pruebas independientemente de las realizadas en el mismo, ya que sirven para establecer parámetros básicos en una fuente de alimentación. Por ejemplo se pueden usar para realizar pruebas con otros transistores, amplificadores, diodos, reguladores, etc.

En el futuro, se podría extender la funcionalidad de dichos códigos en función de las necesidades del usuario, como se viene explicando anteriormente gracias a la flexibilidad de la instrumentación virtual. Además, a pesar de estar perfectamente explicada la funcionalidad de cada módulo en esta memoria, se podría elaborar una interfaz gráfica que haga más intuitivo el uso del código y mejore la interactividad con el usuario.

Apéndice A
Datos exportados de la prueba 2

Código A.1 Archivo test1.py.

```
Vgs = -1.0
Vgs_out = [-1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003,
    -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003, -1.003]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.711, 0.762, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 1.993,
    2.057, 2.107, 2.145, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.6, 0.6]
Vgs = -0.9
Vgs_out = [-0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901,
    -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901, -0.901]
```

```
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.66, 0.711, 0.762, 0.812, 0.851, 0.901, 0.952, 1.003,
     1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447, 1.498,
    1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.942, 1.993, 2.057,
    2.107, 2.145, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552, 2.602,
    2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047, 3.098,
    3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.504]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6]
Vgs = -0.8
Vgs_out = [-0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8,
    -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8,
    -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8,
    -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8,
    -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8,
    -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8, -0.8]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.711, 0.762, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.942, 2.006,
    2.044, 2.107, 2.158, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.504]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, 0.6, -0.5, -0.5, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6]
Vgs = -0.7000000000000001
Vgs_out = [-0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711,
    -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711, -0.711]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.66, 0.698, 0.749, 0.812, 0.851, 0.901, 0.952, 1.003,
     1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447, 1.498,
```

```
    1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 1.993, 2.057,
    2.095, 2.158, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552, 2.602,
    2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047, 3.098,
    3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.6, -0.5, 0.6, 0.6, -0.5, 0.6, 0.6,
    0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 1.7]
Vgs = -0.6000000000000001
Vgs_out = [-0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609,
    -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609,
    -0.609, -0.609, -0.609, -0.609, -0.597, -0.609, -0.597, -0.609, -0.609,
    -0.597, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.597,
    -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609,
    -0.609, -0.609, -0.609, -0.609, -0.609, -0.597, -0.609, -0.609, -0.597,
    -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609,
    -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609, -0.609]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.698, 0.749, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 1.993,
    2.044, 2.107, 2.145, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, 0.6, -0.5, 0.6, -0.5, -0.5, -0.5, 0.6, 0.6, 0.6,
    0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 1.7, 1.7,
    1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7]
Vgs = -0.5000000000000001
Vgs_out = [-0.495, -0.508, -0.495, -0.495, -0.495, -0.495, -0.508, -0.508,
    -0.508, -0.495, -0.495, -0.508, -0.495, -0.495, -0.508, -0.495, -0.508,
    -0.508, -0.495, -0.495, -0.508, -0.495, -0.495, -0.508, -0.495, -0.495,
    -0.508, -0.495, -0.495, -0.495, -0.495, -0.495, -0.495, -0.495, -0.495,
    -0.495, -0.495, -0.495, -0.508, -0.508, -0.508, -0.495, -0.495, -0.508,
    -0.495, -0.508, -0.508, -0.508, -0.495, -0.495, -0.495, -0.495, -0.495,
    -0.508, -0.495, -0.508, -0.495, -0.495, -0.495, -0.495, -0.495, -0.495,
    -0.508, -0.495, -0.508, -0.508, -0.495, -0.495, -0.495, -0.508, -0.495]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.609, 0.647, 0.711, 0.749, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 2.006,
```

```
    2.057, 2.107, 2.158, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5,
    -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, -0.5, 0.6, -0.5, -0.5, 0.6, 0.6,
    0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6,
    0.6, 0.6, 0.6, 0.6, 1.7, 0.6, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7,
    2.8, 2.8, 2.8, 2.8, 2.8, 2.8, 2.8, 2.8, 3.9, 3.9, 3.9, 3.9, 3.9, 3.9, 3.9,
    4.8999999999999995, 4.8999999999999995]
Vgs = -0.40000000000000013
Vgs_out = [-0.394, -0.406, -0.394, -0.406, -0.394, -0.394, -0.406, -0.394,
    -0.394, -0.394, -0.394, -0.394, -0.394, -0.406, -0.406, -0.394, -0.394,
    -0.394, -0.406, -0.394, -0.406, -0.394, -0.394, -0.394, -0.394, -0.394,
    -0.394, -0.394, -0.406, -0.394, -0.394, -0.394, -0.406, -0.394, -0.394,
    -0.394, -0.394, -0.394, -0.406, -0.406, -0.406, -0.406, -0.406, -0.406,
    -0.406, -0.394, -0.406, -0.406, -0.406, -0.406, -0.394, -0.394, -0.406,
    -0.406, -0.406, -0.394, -0.406, -0.394, -0.394, -0.406, -0.394, -0.394,
    -0.394, -0.406, -0.394, -0.394, -0.394, -0.394, -0.394, -0.406, -0.394]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.711, 0.762, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.942, 1.993,
    2.057, 2.095, 2.158, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, -0.5, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6, 0.6,
    0.6, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7, 1.7,
    2.8, 2.8, 2.8, 2.8, 2.8, 2.8, 2.8, 9.2, 11.4, 11.4, 12.5, 13.5, 13.5, 14.6,
     14.6, 15.7, 15.7, 16.8, 16.8, 16.8, 17.9, 18.9, 18.9, 18.9, 20.0, 20.0,
    21.1, 21.1, 22.2, 22.2, 23.2, 23.2, 24.299999999999997, 24.299999999999997,
     24.299999999999997, 25.4, 26.5, 26.5, 26.5, 27.5, 27.5, 28.6, 28.6]
Vgs = -0.30000000000000016
Vgs_out = [-0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292,
    -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292, -0.292]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.711, 0.749, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 1.993,
    2.044, 2.107, 2.145, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
```

```
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, 0.6, 1.7, 1.7, 2.8, 2.8, 2.8, 3.9, 3.9, 3.9, 3.9, 3.9, 3.9,
    4.8999999999999995, 4.8999999999999995, 4.8999999999999995, 6.0, 6.0, 6.0,
    7.1000000000000005, 7.1000000000000005, 8.200000000000001,
    8.200000000000001, 8.200000000000001, 9.2, 9.2, 10.3, 10.3, 10.3, 11.4,
    11.4, 12.5, 12.5, 12.5, 13.5, 13.5, 14.6, 14.6, 15.7, 15.7, 15.7, 16.8,
    16.8, 17.9, 17.9, 18.9, 18.9, 18.9, 20.0, 20.0, 21.1, 21.1, 22.2, 22.2,
    23.2, 23.2, 23.2, 24.299999999999997, 24.299999999999997, 25.4, 25.4, 26.5,
     26.5, 27.5, 27.5, 28.6, 28.6, 29.7, 29.7, 30.8, 30.8]
Vgs = -0.20000000000000018
Vgs_out = [-0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203,
    -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203, -0.203]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.698, 0.762, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 2.006,
    2.057, 2.107, 2.145, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1]
Ids_out = [-0.5, 1.7, 3.9, 4.8999999999999995, 6.0, 7.1000000000000005,
    7.1000000000000005, 8.200000000000001, 8.200000000000001, 9.2, 9.2, 10.3,
    10.3, 11.4, 11.4, 12.5, 13.5, 13.5, 14.6, 14.6, 15.7, 16.8, 17.9, 17.9,
    18.9, 20.0, 20.0, 21.1, 22.2, 22.2, 23.2, 24.299999999999997, 25.4, 26.5,
    26.5, 27.5, 28.6, 28.6, 29.7, 30.8, 30.8, 31.9, 31.9, 32.9, 34.0, 34.0,
    35.1, 35.1, 36.2, 36.2, 37.199999999999996, 37.199999999999996, 38.3, 38.3,
     39.4, 39.4, 40.5, 40.5, 41.5, 41.5, 42.6, 42.6, 42.6, 43.7, 44.8, 44.8,
    44.8, 45.900000000000006, 45.900000000000006, 45.900000000000006, 46.9]
Vgs = -0.1000000000000002
Vgs_out = [-0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102,
    -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102, -0.102]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.597, 0.647, 0.698, 0.762, 0.812, 0.851, 0.901, 0.952,
```

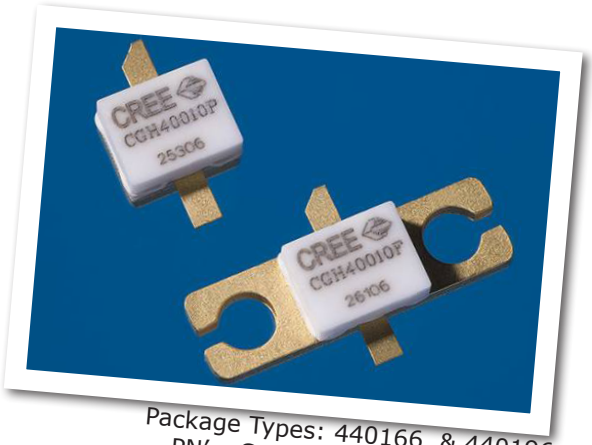
```
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.942, 1.993,
    2.044, 2.107, 2.158, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.491]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -1.2,
    -0.1, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2]
Ids_out = [-0.5, 2.8, 6.0, 9.2, 11.4, 12.5, 13.5, 14.6, 15.7, 15.7, 15.7, 15.7,
     15.7, 16.8, 16.8, 17.9, 17.9, 18.9, 18.9, 20.0, 21.1, 21.1, 22.2, 22.2,
    23.2, 24.299999999999997, 24.299999999999997, 25.4, 26.5, 26.5, 27.5, 28.6,
     29.7, 29.7, 30.8, 30.8, 31.9, 32.9, 32.9, 34.0, 35.1, 35.1, 36.2, 36.2,
    37.199999999999996, 37.199999999999996, 38.3, 38.3, 39.4, 39.4, 40.5, 41.5,
     41.5, 41.5, 42.6, 43.7, 43.7, 43.7, 44.8, 44.8, 45.900000000000006,
    45.900000000000006, 46.9, 46.9, 48.0, 48.0, 48.0, 49.099999999999994,
    49.099999999999994, 49.099999999999994, 50.2]
Vgs = -2.220446049250313e-16
Vgs_out = [-0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
    -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
    -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
    -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
    -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0,
    -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0, -0.0]
Vds_out = [0.0, 0.051, 0.102, 0.152, 0.203, 0.254, 0.292, 0.343, 0.394, 0.444,
    0.495, 0.546, 0.609, 0.647, 0.698, 0.762, 0.812, 0.851, 0.901, 0.952,
    1.003, 1.054, 1.104, 1.155, 1.206, 1.257, 1.308, 1.346, 1.396, 1.447,
    1.498, 1.549, 1.6, 1.65, 1.701, 1.752, 1.803, 1.841, 1.892, 1.955, 2.006,
    2.057, 2.107, 2.158, 2.209, 2.26, 2.31, 2.349, 2.399, 2.45, 2.501, 2.552,
    2.602, 2.653, 2.704, 2.755, 2.806, 2.844, 2.894, 2.945, 2.996, 3.047,
    3.098, 3.148, 3.199, 3.25, 3.301, 3.351, 3.39, 3.44, 3.504]
Igs_out = [-0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1, -0.1,
    -0.1, -0.1, -0.1, -0.1, -1.2, -0.1, -0.1, -1.2, -0.1, -1.2, -1.2, -1.2,
    -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2, -1.2]
Ids_out = [-0.5, 3.9, 7.1000000000000005, 11.4, 14.6, 16.8, 18.9, 21.1, 22.2,
    24.299999999999997, 24.299999999999997, 24.299999999999997,
    24.299999999999997, 24.299999999999997, 24.299999999999997, 23.2, 23.2,
    24.299999999999997, 24.299999999999997, 24.299999999999997, 25.4, 25.4,
    26.5, 27.5, 27.5, 28.6, 29.7, 29.7, 29.7, 30.8, 30.8, 31.9, 32.9, 32.9,
    34.0, 35.1, 35.1, 36.2, 36.2, 37.199999999999996, 38.3, 38.3, 39.4, 39.4,
    40.5, 40.5, 41.5, 42.6, 42.6, 43.7, 43.7, 44.8, 44.8, 45.900000000000006,
    45.900000000000006, 46.9, 46.9, 46.9, 48.0, 48.0, 49.099999999999994,
    49.099999999999994, 49.099999999999994, 50.2, 50.2, 51.2, 51.2, 51.2, 51.2,
     52.3, 52.3]
```

Apéndice B
Datasheet del amplificador CGH40010

CGH40010

10 W, RF Power GaN HEMT

Cree's CGH40010 is an unmatched, gallium nitride (GaN) high electron mobility transistor (HEMT). The CGH40010, operating from a 28 volt rail, offers a general purpose, broadband solution to a variety of RF and microwave applications. GaN HEMTs offer high efficiency, high gain and wide bandwidth capabilities making the CGH40010 ideal for linear and compressed amplifier circuits. The transistor is available in both screw-down, flange and solder-down, pill packages.



Package Types: 440166, & 440196
PN's: CGH40010F & CGH40010P

FEATURES

- Up to 6 GHz Operation
- 16 dB Small Signal Gain at 2.0 GHz
- 14 dB Small Signal Gain at 4.0 GHz
- 13 W typical P_{SAT}
- 65 % Efficiency at P_{SAT}
- 28 V Operation

APPLICATIONS

- 2-Way Private Radio
- Broadband Amplifiers
- Cellular Infrastructure
- Test Instrumentation
- Class A, AB, Linear amplifiers suitable for OFDM, W-CDMA, EDGE, CDMA waveforms

RoHS COMPLIANT

Large Signal Models Available for SiC & GaN

Absolute Maximum Ratings (not simultaneous) at 25˚C Case Temperature

Parameter	Symbol	Rating	Units
Drain-Source Voltage	V_{DSS}	84	Volts
Gate-to-Source Voltage	V_{GS}	-10, +2	Volts
Storage Temperature	T_{STG}	-65, +150	˚C
Operating Junction Temperature	T_J	225	˚C
Maximum Forward Gate Current	I_{GMAX}	4.0	mA
Soldering Temperature[1]	T_S	245	˚C
Screw Torque	τ	60	in-oz
Thermal Resistance, Junction to Case[2]	$R_{\theta JC}$	8.0	˚C/W
Case Operating Temperature[2,3]	T_C	-40, +150	˚C

Note:
[1] Refer to the Application Note on soldering at www.cree.com/products/wireless_appnotes.asp
[2] Measured for the CGH40010F at P_{DISS} = 14 W.
[3] See also, the Power Dissipation De-rating Curve on Page 6.

Electrical Characteristics (T_C = 25˚C)

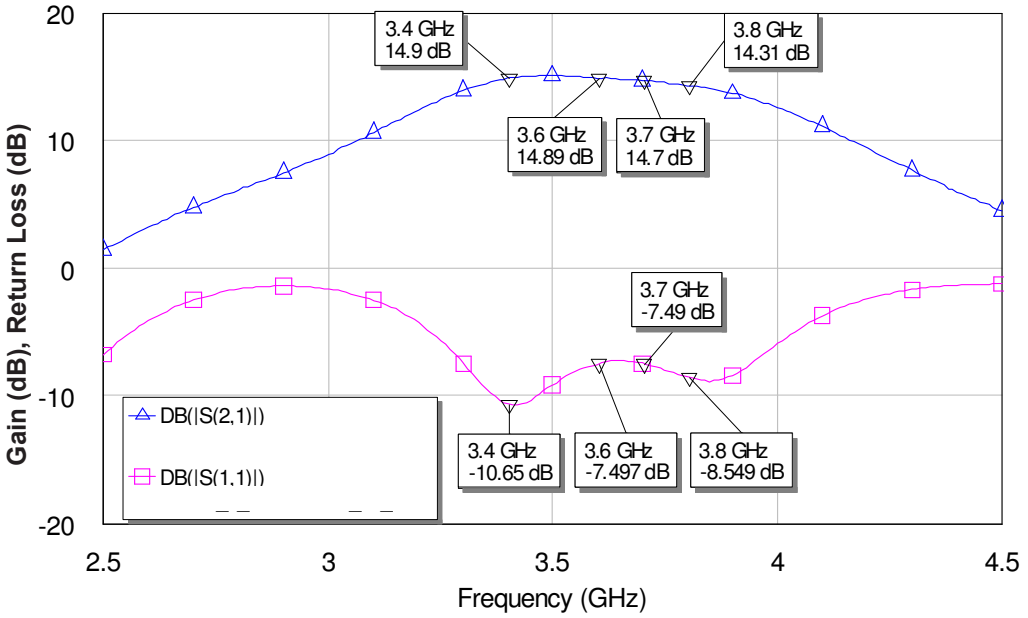
Characteristics	Symbol	Min.	Typ.	Max.	Units	Conditions
DC Characteristics[1]						
Gate Threshold Voltage	$V_{GS(th)}$	-3.8	-3.3	-2.3	V_{DC}	V_{DS} = 10 V, I_D = 3.6 mA
Gate Quiescent Voltage	$V_{GS(Q)}$	–	-3.0	–	V_{DC}	V_{DS} = 28 V, I_D = 200 mA
Saturated Drain Current	I_{DS}	2.9	3.5	–	A	V_{DS} = 6.0 V, V_{GS} = 2.0 V
Drain-Source Breakdown Voltage	V_{BR}	120	–	–	V_{DC}	V_{GS} = -8 V, I_D = 3.6 mA
RF Characteristics[2] (T_C = 25˚C, F_0 = 3.7 GHz unless otherwise noted)						
Small Signal Gain	G_{SS}	12.5	14.5	–	dB	V_{DD} = 28 V, I_{DQ} = 200 mA
Power Output[3]	P_{SAT}	10	12.5	–	W	V_{DD} = 28 V, I_{DQ} = 200 mA
Drain Efficiency[4]	η	55	65	–	%	V_{DD} = 28 V, I_{DQ} = 200 mA, P_{SAT}
Output Mismatch Stress	VSWR	–	–	10 : 1	Ψ	No damage at all phase angles, V_{DD} = 28 V, I_{DQ} = 200 mA, P_{OUT} = 10 W CW
Dynamic Characteristics						
Input Capacitance	C_{GS}	–	4.5	–	pF	V_{DS} = 28 V, V_{gs} = -8 V, f = 1 MHz
Output Capacitance	C_{DS}	–	1.3	–	pF	V_{DS} = 28 V, V_{gs} = -8 V, f = 1 MHz
Feedback Capacitance	C_{GD}	–	0.2	–	pF	V_{DS} = 28 V, V_{gs} = -8 V, f = 1 MHz

Notes:
[1] Measured on wafer prior to packaging.
[2] Measured in CGH40010-TB.
[3] P_{SAT} is defined as I_G = 0.36 mA.
[4] Drain Efficiency = P_{OUT} / P_{DC}

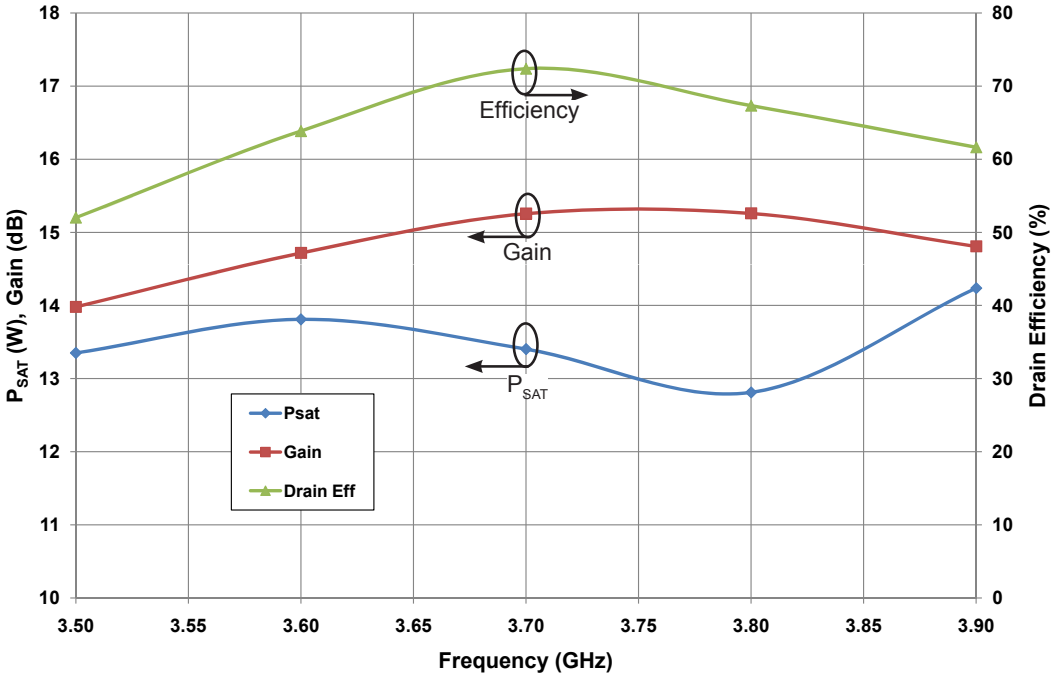
Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Typical Performance

Small Signal Gain and Return Loss vs Frequency of the CGH40010 in the CGH40010-TB



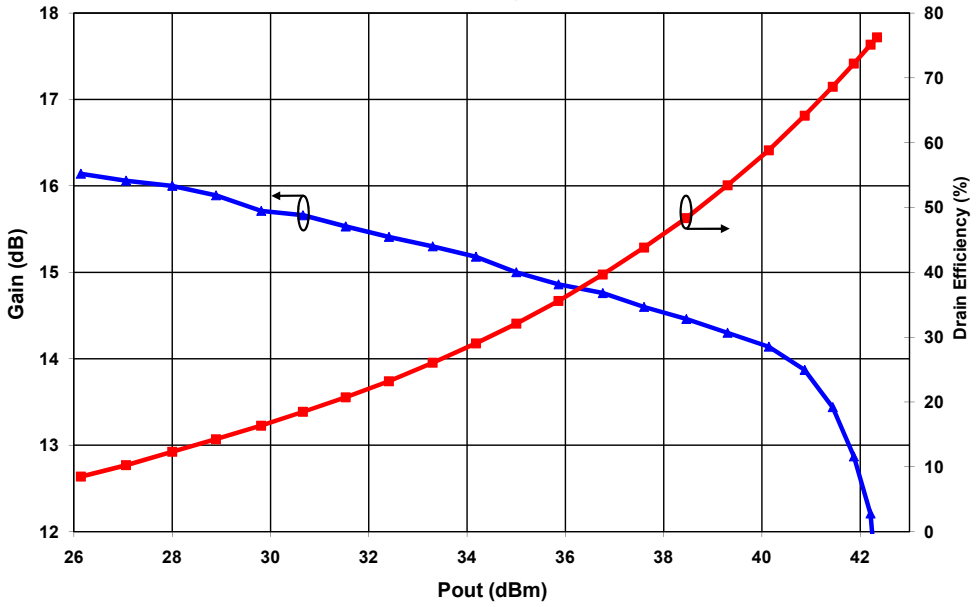
P_{SAT}, Gain, and Drain Efficiency vs Frequency of the CGH40010F in the CGH40010-TB
V_{DD} = 28 V, I_{DQ} = 200 mA



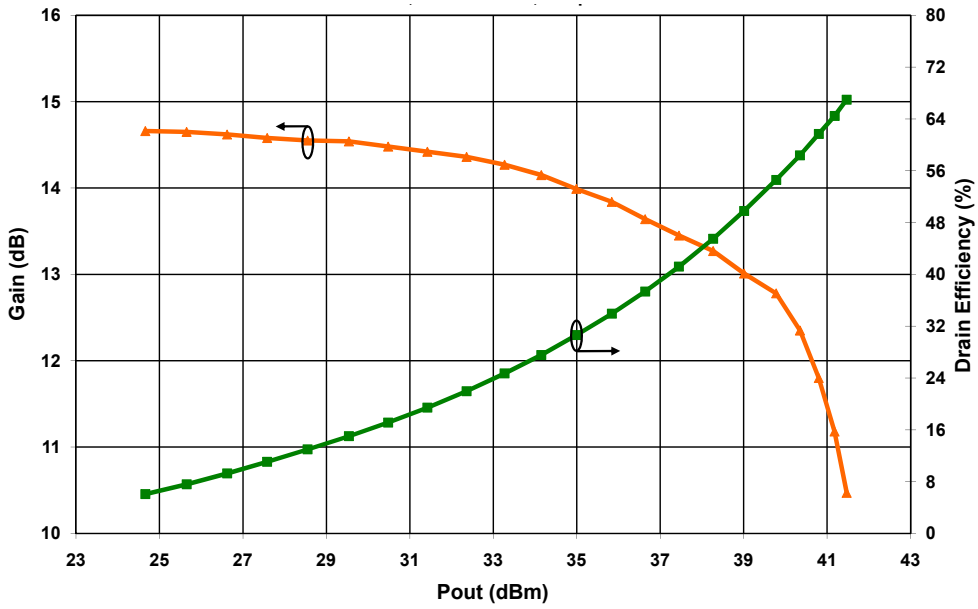
Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Typical Performance

Swept CW Data of CGH40010F vs. Output Power with Source and Load Impedances Optimized for Drain Efficiency at 2.0 GHz
V_{DD} = 28 V, I_{DQ} = 200 mA



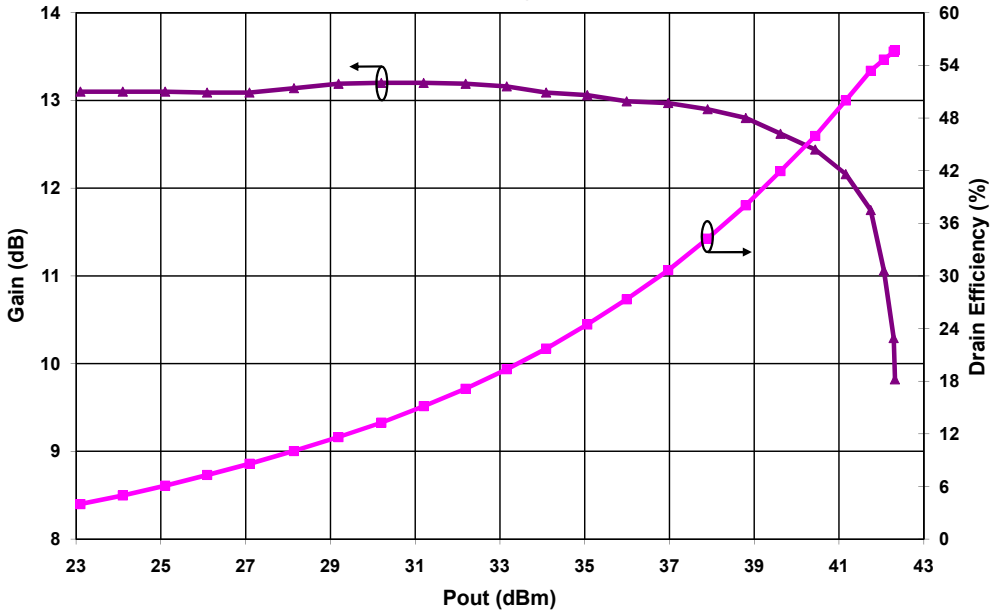
Swept CW Data of CGH40010F vs. Output Power with Source and Load Impedances Optimized for Drain Efficiency at 3.6 GHz
V_{DD} = 28 V, I_{DQ} = 200 mA



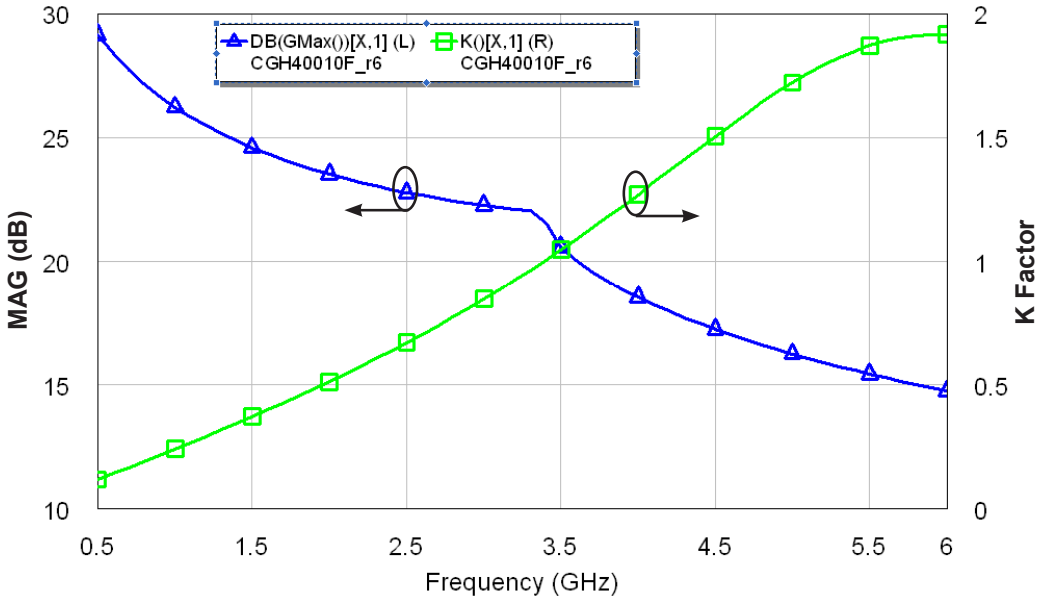
Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Typical Performance

Swept CW Data of CGH40010F vs. Output Power with Source and Load Impedances Optimized for P1 Power at 3.6 GHz
V_{DD} = 28 V, I_{DQ} = 200 mA



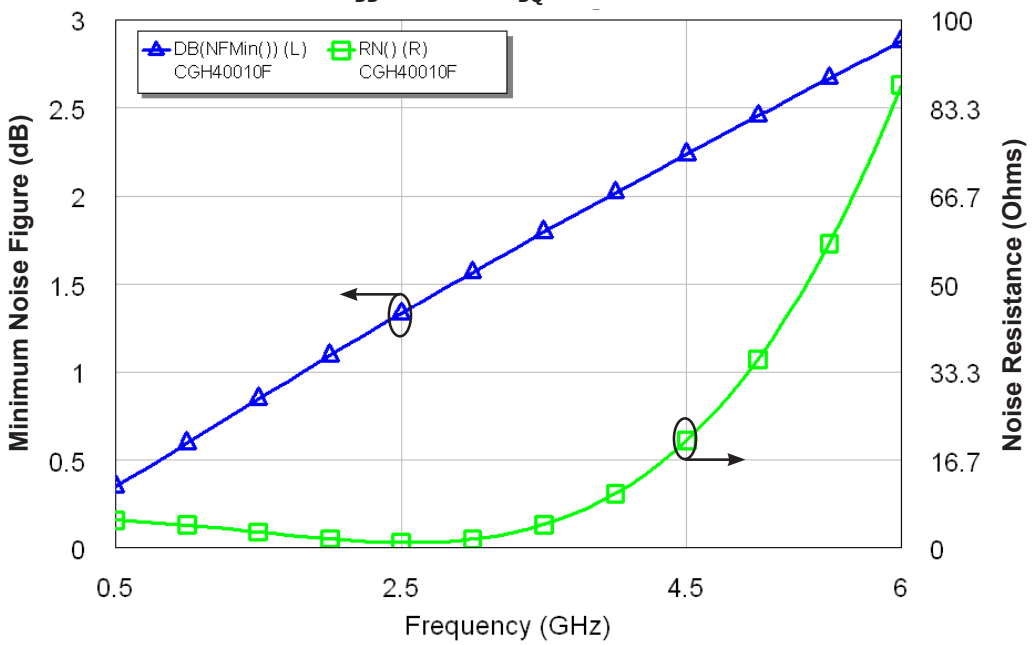
Simulated Maximum Available Gain and K Factor of the CGH40010F
V_{DD} = 28 V, I_{DQ} = 200 mA



Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Typical Noise Performance

Simulated Minimum Noise Figure and Noise Resistance vs Frequency of the CGH40010F
V_{DD} = 28 V, I_{DQ} = 100 mA

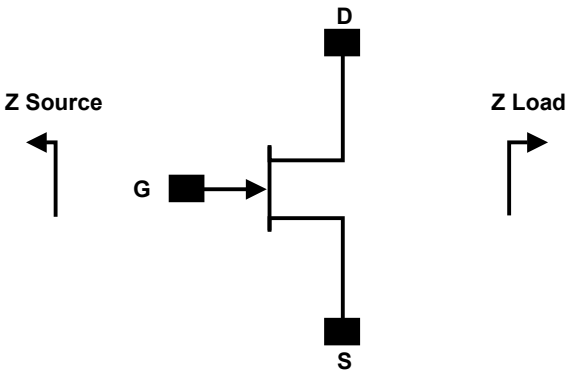


Electrostatic Discharge (ESD) Classifications

Parameter	Symbol	Class	Test Methodology
Human Body Model	HBM	1A > 250 V	JEDEC JESD22 A114-D
Charge Device Model	CDM	1 < 200 V	JEDEC JESD22 C101-C

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

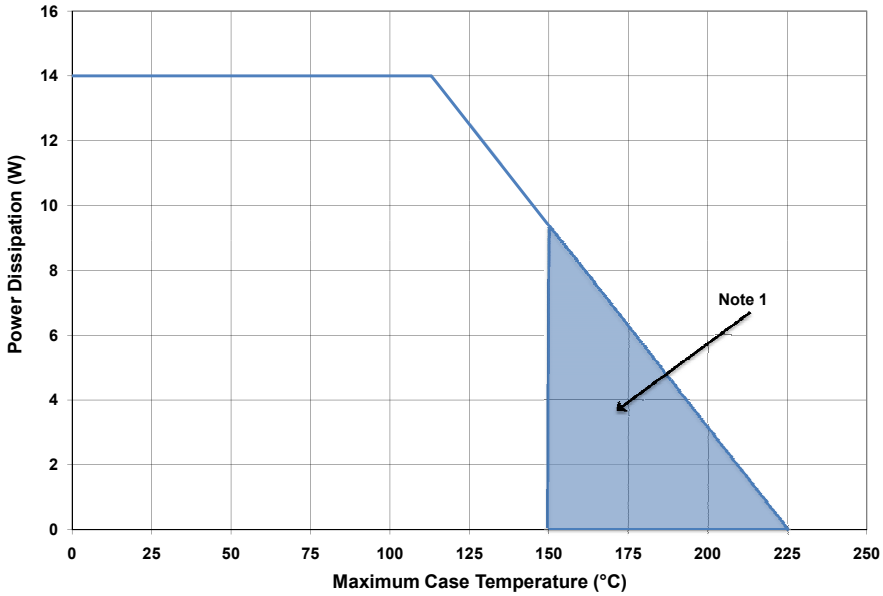
Source and Load Impedances



Frequency (MHz)	Z Source	Z Load
500	20.2 + j16.18	51.7 + j15.2
1000	8.38 + j9.46	41.4 + j28.5
1500	7.37 + j0	28.15 + j29
2500	3.19 - j4.76	19 + j9.2
3500	3.18 - j13.3	14.6 + j7.46

Note 1. V_{DD} = 28V, I_{DQ} = 200mA in the 440166 package.
Note 2. Optimized for power, gain, P_{SAT} and PAE.
Note 3. When using this device at low frequency, series resistors should be used to maintain amplifier stability.

CGH40010 Power Dissipation De-rating Curve



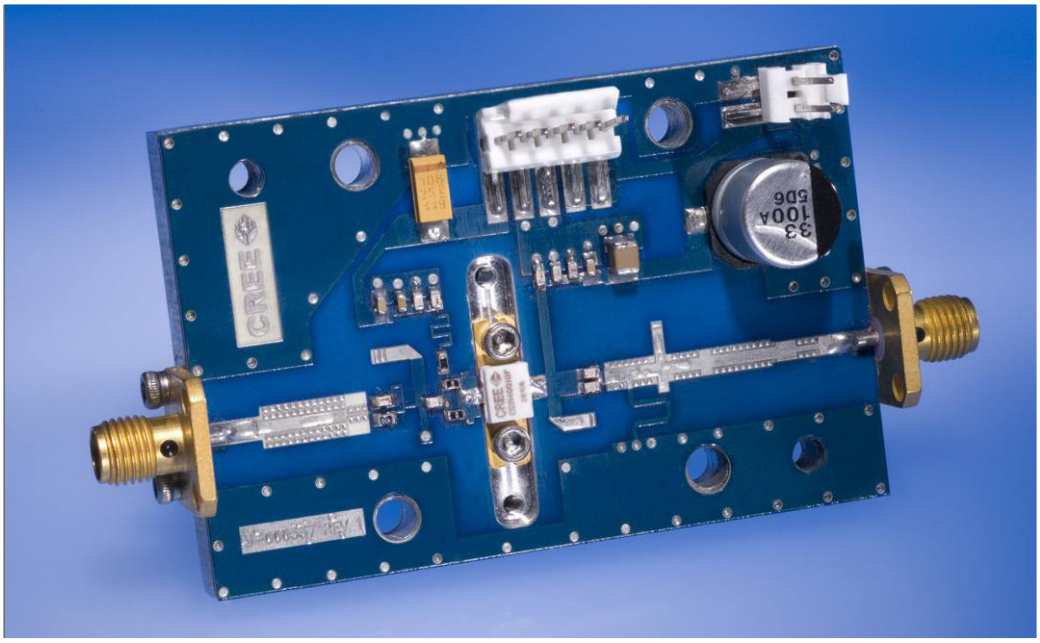
Note 1. Area exceeds Maximum Case Operating Temperature (See Page 2).

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

CGH40010-TB Demonstration Amplifier Circuit Bill of Materials

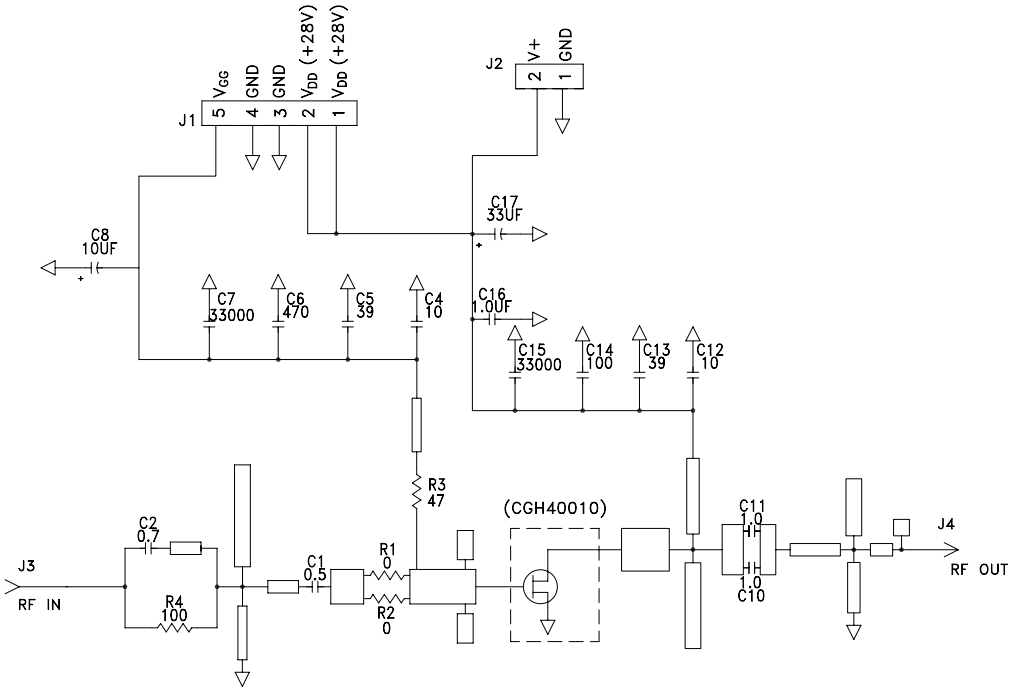
Designator	Description	Qty
R1,R2	RES,1/16W,0603,1%,0 OHMS	1
R3	RES,1/16W,0603,1%,47 OHMS	1
R4	RES,1/16W,0603,1%,100 OHMS	1
C6	CAP, 470PF, 5%,100V, 0603	1
C17	CAP, 33 UF, 20%, G CASE	1
C16	CAP, 1.0UF, 100V, 10%, X7R, 1210	1
C8	CAP 10UF 16V TANTALUM	1
C14	CAP, 100.0pF, +/-5%, 0603	1
C1	CAP, 0.5pF, +/-0.05pF, 0603	1
C2	CAP, 0.7pF, +/-0.1pF, 0603	1
C10,C11	CAP, 1.0pF, +/-0.1pF, 0603	2
C4,C12	CAP, 10.0pF,+/-5%, 0603	2
C5,C13	CAP, 39pF, +/-5%, 0603	2
C7,C15	CAP,33000PF, 0805,100V, X7R	2
J3,J4	CONN SMA STR PANEL JACK RECP	1
J2	HEADER RT>PLZ.1CEN LK 2 POS	1
J1	HEADER RT>PLZ .1CEN LK 5POS	1
-	PCB, RO4350B, Er = 3.48, h = 20 mil	1
Q1	CGH40010F or CGH40010P	1

CGH40010-TB Demonstration Amplifier Circuit

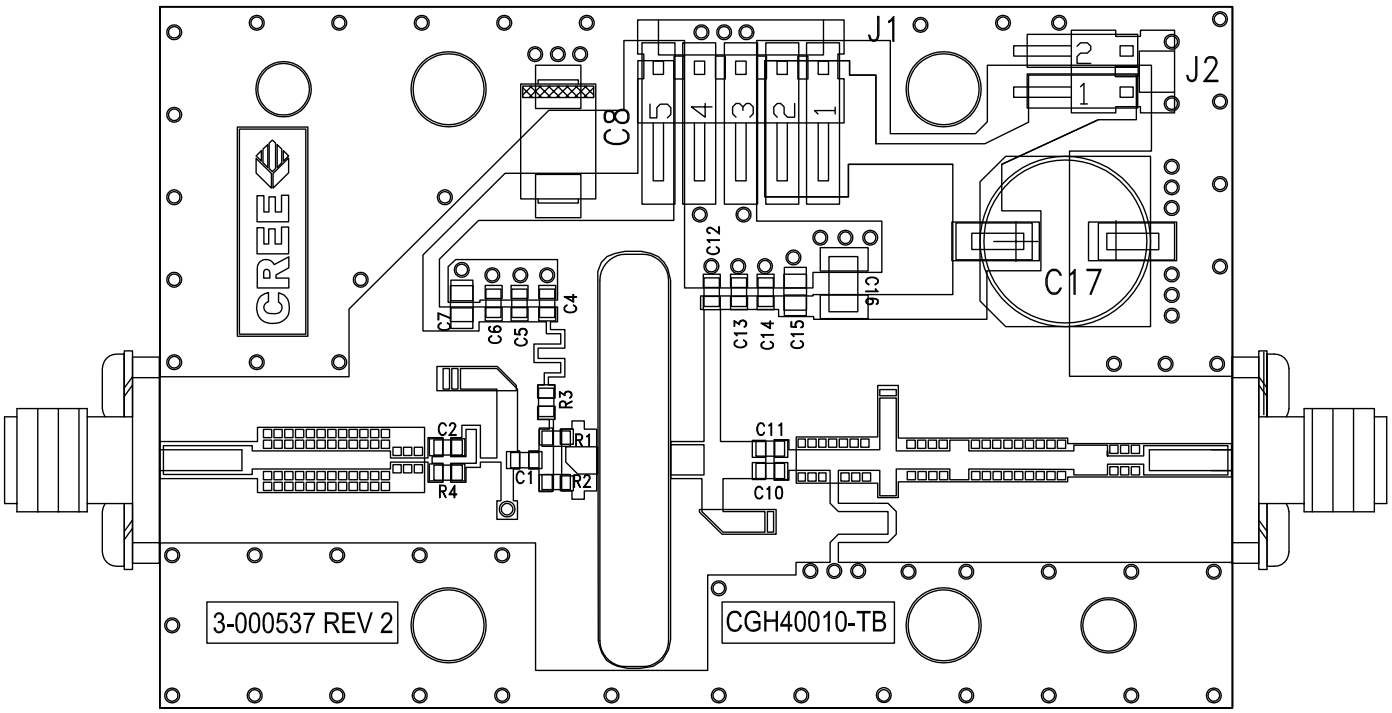


Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

CGH40010-TB Demonstration Amplifier Circuit Schematic



CGH40010-TB Demonstration Amplifier Circuit Outline



Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Typical Package S-Parameters for CGH40010
(Small Signal, V_{DS} = 28 V, I_{DQ} = 100 mA, angle in degrees)

Frequency	Mag S11	Ang S11	Mag S21	Ang S21	Mag S12	Ang S12	Mag S22	Ang S22
500 MHz	0.909	-123.34	17.19	108.22	0.027	21.36	0.343	-90.81
600 MHz	0.902	-133.06	14.86	101.82	0.028	15.60	0.329	-98.65
700 MHz	0.897	-140.73	13.04	96.45	0.028	10.87	0.321	-104.84
800 MHz	0.894	-146.96	11.58	91.78	0.029	6.84	0.317	-109.84
900 MHz	0.891	-152.16	10.41	87.61	0.029	3.33	0.316	-113.95
1.0 GHz	0.890	-156.60	9.43	83.82	0.029	0.19	0.318	-117.42
1.1 GHz	0.889	-160.47	8.62	80.31	0.029	-2.66	0.321	-120.40
1.2 GHz	0.888	-163.90	7.93	77.02	0.029	-5.28	0.326	-123.02
1.3 GHz	0.887	-166.99	7.34	73.90	0.029	-7.72	0.332	-125.36
1.4 GHz	0.887	-169.80	6.82	70.92	0.029	-10.01	0.338	-127.51
1.5 GHz	0.887	-172.39	6.38	68.05	0.029	-12.18	0.345	-129.50
1.6 GHz	0.887	-174.80	5.98	65.28	0.028	-14.24	0.353	-131.37
1.7 GHz	0.887	-177.07	5.63	62.59	0.028	-16.21	0.360	-133.15
1.8 GHz	0.887	-179.22	5.32	59.97	0.028	-18.09	0.369	-134.87
1.9 GHz	0.887	178.73	5.04	57.41	0.028	-19.91	0.377	-136.54
2.0 GHz	0.888	176.76	4.78	54.89	0.027	-21.66	0.385	-138.17
2.1 GHz	0.888	174.86	4.55	52.42	0.027	-23.35	0.393	-139.77
2.2 GHz	0.888	173.02	4.34	49.99	0.027	-24.98	0.402	-141.34
2.3 GHz	0.888	171.23	4.15	47.60	0.026	-26.56	0.410	-142.90
2.4 GHz	0.889	169.48	3.97	45.24	0.026	-28.08	0.418	-144.45
2.5 GHz	0.889	167.76	3.81	42.90	0.026	-29.55	0.426	-145.99
2.6 GHz	0.890	166.07	3.66	40.59	0.025	-30.98	0.434	-147.53
2.7 GHz	0.890	164.39	3.53	38.30	0.025	-32.36	0.442	-149.06
2.8 GHz	0.890	162.74	3.40	36.03	0.025	-33.69	0.450	-150.59
2.9 GHz	0.891	161.10	3.28	33.78	0.024	-34.97	0.458	-152.12
3.0 GHz	0.891	159.46	3.17	31.55	0.024	-36.20	0.465	-153.65
3.2 GHz	0.892	156.21	2.97	27.12	0.023	-38.51	0.479	-156.72
3.4 GHz	0.893	152.96	2.79	22.73	0.022	-40.63	0.493	-159.80
3.6 GHz	0.893	149.69	2.64	18.38	0.022	-42.52	0.505	-162.90
3.8 GHz	0.894	146.38	2.50	14.05	0.021	-44.17	0.517	-166.03
4.0 GHz	0.894	143.03	2.38	9.72	0.020	-45.56	0.527	-169.19
4.2 GHz	0.894	139.61	2.28	5.40	0.019	-46.67	0.537	-172.39
4.4 GHz	0.895	136.11	2.18	1.07	0.019	-47.46	0.546	-175.64
4.6 GHz	0.895	132.53	2.09	-3.29	0.018	-47.90	0.554	-178.95
4.8 GHz	0.895	128.85	2.01	-7.68	0.017	-47.96	0.561	177.69
5.0 GHz	0.895	125.06	1.94	-12.10	0.017	-47.61	0.568	174.25
5.2 GHz	0.895	121.15	1.88	-16.58	0.016	-46.84	0.573	170.72
5.4 GHz	0.895	117.11	1.82	-21.12	0.016	-45.67	0.578	167.10
5.6 GHz	0.895	112.94	1.77	-25.73	0.015	-44.12	0.582	163.38
5.8 GHz	0.895	108.62	1.72	-30.42	0.015	-42.30	0.586	159.54
6.0 GHz	0.895	104.15	1.68	-35.20	0.015	-40.33	0.589	155.56

Download this s-parameter file in ".s2p" format at http://www.cree.com/products/wireless_s-parameters.asp

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Typical Package S-Parameters for CGH40010
(Small Signal, V_{DS} = 28 V, I_{DQ} = 200 mA, angle in degrees)

Frequency	Mag S11	Ang S11	Mag S21	Ang S21	Mag S12	Ang S12	Mag S22	Ang S22
500 MHz	0.911	-130.62	18.41	105.41	0.022	19.44	0.303	-112.24
600 MHz	0.906	-139.65	15.80	99.47	0.023	14.31	0.299	-119.83
700 MHz	0.902	-146.70	13.80	94.50	0.023	10.17	0.298	-125.50
800 MHz	0.899	-152.41	12.22	90.19	0.023	6.68	0.299	-129.85
900 MHz	0.898	-157.17	10.96	86.34	0.024	3.67	0.302	-133.28
1.0 GHz	0.896	-161.24	9.92	82.82	0.024	0.99	0.305	-136.05
1.1 GHz	0.896	-164.79	9.06	79.56	0.024	-1.41	0.309	-138.34
1.2 GHz	0.895	-167.95	8.33	76.49	0.024	-3.62	0.314	-140.30
1.3 GHz	0.895	-170.80	7.70	73.57	0.023	-5.66	0.320	-142.01
1.4 GHz	0.894	-173.41	7.17	70.78	0.023	-7.56	0.326	-143.54
1.5 GHz	0.894	-175.82	6.70	68.08	0.023	-9.35	0.332	-144.94
1.6 GHz	0.894	-178.09	6.28	65.47	0.023	-11.05	0.338	-146.24
1.7 GHz	0.894	179.78	5.92	62.92	0.023	-12.66	0.345	-147.48
1.8 GHz	0.894	177.75	5.59	60.43	0.023	-14.19	0.352	-148.68
1.9 GHz	0.894	175.81	5.30	57.99	0.023	-15.65	0.358	-149.84
2.0 GHz	0.894	173.94	5.04	55.59	0.022	-17.05	0.365	-150.99
2.1 GHz	0.894	172.13	4.80	53.23	0.022	-18.39	0.372	-152.12
2.2 GHz	0.894	170.37	4.58	50.91	0.022	-19.67	0.379	-153.26
2.3 GHz	0.895	168.65	4.38	48.61	0.022	-20.90	0.386	-154.39
2.4 GHz	0.895	166.96	4.20	46.33	0.021	-22.08	0.393	-155.54
2.5 GHz	0.895	165.30	4.03	44.08	0.021	-23.20	0.400	-156.69
2.6 GHz	0.895	163.66	3.88	41.84	0.021	-24.27	0.407	-157.85
2.7 GHz	0.895	162.04	3.74	39.63	0.021	-25.28	0.414	-159.03
2.8 GHz	0.895	160.43	3.60	37.43	0.020	-26.25	0.420	-160.22
2.9 GHz	0.896	158.83	3.48	35.24	0.020	-27.16	0.427	-161.42
3.0 GHz	0.896	157.24	3.37	33.06	0.020	-28.02	0.433	-162.64
3.2 GHz	0.896	154.06	3.16	28.74	0.019	-29.57	0.446	-165.13
3.4 GHz	0.896	150.87	2.98	24.44	0.019	-30.88	0.457	-167.69
3.6 GHz	0.896	147.66	2.82	20.16	0.018	-31.95	0.468	-170.31
3.8 GHz	0.897	144.41	2.68	15.89	0.018	-32.76	0.478	-173.00
4.0 GHz	0.897	141.10	2.56	11.61	0.017	-33.30	0.488	-175.77
4.2 GHz	0.897	137.72	2.45	7.33	0.017	-33.55	0.497	-178.61
4.4 GHz	0.897	134.26	2.35	3.03	0.017	-33.50	0.505	178.47
4.6 GHz	0.897	130.71	2.26	-1.31	0.016	-33.18	0.512	175.46
4.8 GHz	0.896	127.06	2.17	-5.68	0.016	-32.58	0.518	172.36
5.0 GHz	0.896	123.30	2.10	-10.09	0.016	-31.74	0.524	169.16
5.2 GHz	0.896	119.42	2.04	-14.57	0.016	-30.72	0.529	165.86
5.4 GHz	0.896	115.41	1.98	-19.10	0.016	-29.60	0.534	162.44
5.6 GHz	0.896	111.26	1.92	-23.71	0.016	-28.46	0.537	158.89
5.8 GHz	0.895	106.97	1.87	-28.40	0.017	-27.41	0.540	155.20
6.0 GHz	0.895	102.53	1.82	-33.19	0.017	-26.54	0.543	151.36

Download this s-parameter file in ".s2p" format at http://www.cree.com/products/wireless_s-parameters.asp

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

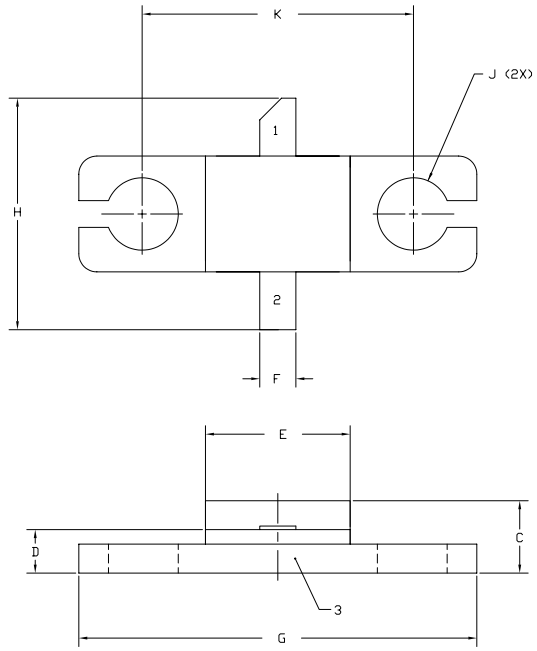
Typical Package S-Parameters for CGH40010
(Small Signal, V_{DS} = 28 V, I_{DQ} = 500 mA, angle in degrees)

Frequency	Mag S11	Ang S11	Mag S21	Ang S21	Mag S12	Ang S12	Mag S22	Ang S22
500 MHz	0.914	-135.02	18.58	103.70	0.020	18.36	0.300	-126.80
600 MHz	0.909	-143.57	15.88	98.05	0.020	13.67	0.302	-133.51
700 MHz	0.906	-150.23	13.83	93.33	0.021	9.90	0.304	-138.40
800 MHz	0.904	-155.61	12.23	89.23	0.021	6.77	0.307	-142.08
900 MHz	0.903	-160.09	10.95	85.56	0.021	4.08	0.311	-144.94
1.0 GHz	0.902	-163.93	9.91	82.21	0.021	1.71	0.314	-147.23
1.1 GHz	0.901	-167.29	9.04	79.09	0.021	-0.41	0.319	-149.10
1.2 GHz	0.901	-170.29	8.31	76.15	0.021	-2.35	0.323	-150.69
1.3 GHz	0.900	-173.00	7.69	73.35	0.021	-4.12	0.328	-152.07
1.4 GHz	0.900	-175.50	7.15	70.66	0.021	-5.78	0.333	-153.29
1.5 GHz	0.900	-177.81	6.69	68.07	0.021	-7.32	0.338	-154.41
1.6 GHz	0.900	-179.98	6.27	65.54	0.021	-8.77	0.344	-155.44
1.7 GHz	0.900	177.96	5.91	63.08	0.020	-10.15	0.349	-156.43
1.8 GHz	0.899	176.00	5.59	60.67	0.020	-11.45	0.355	-157.38
1.9 GHz	0.899	174.12	5.30	58.30	0.020	-12.68	0.361	-158.30
2.0 GHz	0.899	172.31	5.04	55.97	0.020	-13.85	0.366	-159.22
2.1 GHz	0.899	170.54	4.80	53.67	0.020	-14.96	0.372	-160.14
2.2 GHz	0.900	168.83	4.58	51.40	0.020	-16.01	0.378	-161.06
2.3 GHz	0.900	167.15	4.39	49.16	0.019	-17.01	0.384	-161.99
2.4 GHz	0.900	165.49	4.21	46.94	0.019	-17.95	0.390	-162.93
2.5 GHz	0.900	163.87	4.04	44.73	0.019	-18.85	0.396	-163.88
2.6 GHz	0.900	162.26	3.89	42.54	0.019	-19.69	0.402	-164.86
2.7 GHz	0.900	160.66	3.75	40.37	0.019	-20.48	0.407	-165.85
2.8 GHz	0.900	159.08	3.62	38.21	0.019	-21.21	0.413	-166.86
2.9 GHz	0.900	157.51	3.50	36.05	0.018	-21.89	0.418	-167.89
3.0 GHz	0.900	155.93	3.39	33.91	0.018	-22.52	0.424	-168.95
3.2 GHz	0.900	152.79	3.18	29.65	0.018	-23.61	0.435	-171.12
3.4 GHz	0.900	149.64	3.00	25.40	0.017	-24.48	0.445	-173.38
3.6 GHz	0.900	146.45	2.85	21.17	0.017	-25.11	0.454	-175.73
3.8 GHz	0.900	143.23	2.71	16.93	0.017	-25.51	0.463	-178.17
4.0 GHz	0.900	139.94	2.58	12.69	0.017	-25.67	0.471	179.30
4.2 GHz	0.900	136.58	2.47	8.43	0.016	-25.60	0.479	176.67
4.4 GHz	0.899	133.14	2.38	4.15	0.016	-25.32	0.486	173.94
4.6 GHz	0.899	129.61	2.29	-0.17	0.016	-24.85	0.492	171.12
4.8 GHz	0.899	125.97	2.21	-4.53	0.016	-24.24	0.498	168.18
5.0 GHz	0.898	122.23	2.13	-8.94	0.016	-23.54	0.503	165.13
5.2 GHz	0.898	118.36	2.07	-13.41	0.016	-22.80	0.507	161.96
5.4 GHz	0.898	114.36	2.01	-17.95	0.017	-22.11	0.511	158.66
5.6 GHz	0.897	110.22	1.95	-22.56	0.017	-21.54	0.514	155.22
5.8 GHz	0.897	105.94	1.90	-27.26	0.018	-21.16	0.517	151.63
6.0 GHz	0.897	101.51	1.86	-32.04	0.019	-21.04	0.519	147.87

Download this s-parameter file in ".s2p" format at http://www.cree.com/products/wireless_s-parameters.asp

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Product Dimensions CGH40010F (Package Type — 440166)



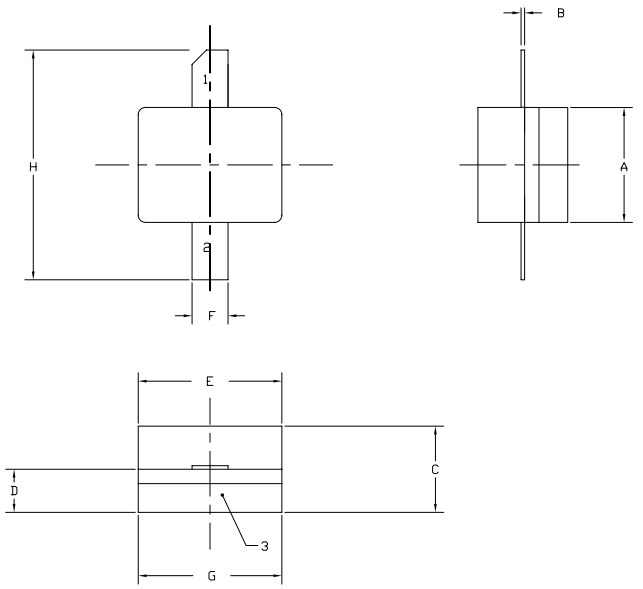
NOTES:

1. DIMENSIONING AND TOLERANICING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. ADHESIVE FROM LID MAY EXTEND A MAXIMUM OF 0.020" BEYOND EDGE OF LID.
4. LID MAY BE MISALIGNED TO THE BODY OF THE PACKAGE BY A MAXIMUM OF 0.008" IN ANY DIRECTION.
5. ALL PLATED SURFACES ARE NI/AU

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.155	0.165	3.94	4.19
B	0.004	0.006	0.10	0.15
C	0.115	0.135	2.92	3.43
D	0.057	0.067	1.45	1.70
E	0.195	0.205	4.95	5.21
F	0.045	0.055	1.14	1.40
G	0.545	0.555	13.84	14.09
H	0.280	0.360	7.87	8.38
J	ø .100		2.54	
K	0.375		9.53	

PIN 1. GATE
PIN 2. DRAIN
PIN 3. SOURCE

Product Dimensions CGH40010P (Package Type — 440196)



NOTES:

1. DIMENSIONING AND TOLERANICING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. ADHESIVE FROM LID MAY EXTEND A MAXIMUM OF 0.020" BEYOND EDGE OF LID.
4. LID MAY BE MISALIGNED TO THE BODY OF THE PACKAGE BY A MAXIMUM OF 0.008" IN ANY DIRECTION.
5. ALL PLATED SURFACES ARE NI/AU

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.155	0.165	3.94	4.19
B	0.003	0.006	0.10	0.15
C	0.115	0.135	2.92	3.17
D	0.057	0.067	1.45	1.70
E	0.195	0.205	4.95	5.21
F	0.045	0.055	1.14	1.40
G	0.195	0.205	4.95	5.21
H	0.280	0.360	7.112	9.114

PIN 1. GATE
PIN 2. DRAIN
PIN 3. SOURCE

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Disclaimer

Specifications are subject to change without notice. Cree, Inc. believes the information contained within this data sheet to be accurate and reliable. However, no responsibility is assumed by Cree for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Cree. Cree makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose. "Typical" parameters are the average values expected by Cree in large quantities and are provided for information purposes only. These values can and do vary in different applications and actual performance can vary over time. All operating parameters should be validated by customer's technical experts for each application. Cree products are not designed, intended or authorized for use as components in applications intended for surgical implant into the body or to support or sustain life, in applications in which the failure of the Cree product could result in personal injury or death or in applications for planning, construction, maintenance or direct operation of a nuclear facility.

For more information, please contact:

Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
www.cree.com/wireless

Ryan Baker
Marketing
Cree, RF Components
919.407.7816

Tom Dekker
Sales Director
Cree, RF Components
919.407.5639

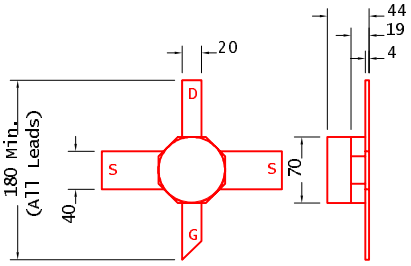
Cree, Inc.
4600 Silicon Drive
Durham, NC 27703
USA Tel: +1.919.313.5300
Fax: +1.919.869.2733
www.cree.com/wireless

Apéndice C
Datasheet del transistor EPB018A5-70

Excelics EPB018A5/A7/A9-70

DATA SHEET
Super Low Noise High Gain Heterojunction FET

- **NON-HERMETIC LOW COST CERAMIC 70 mil PACKAGE**
- **TYPICAL 0.50~0.90dB NOISE FIGURE AND 11.5~13.0dB ASSOCIATED GAIN AT 12GHz**
- **0.3 X 180 MICRON RECESSED " MUSHROOM" GATE**
- **Si_3N_4 PASSIVATION**
- **ADVANCED EPITAXIAL HETEROJUNCTION PROFILE PROVIDES SUPER LOW NOISE, HIGH GAIN AND HIGH RELIABILITY**



All Dimensions In mils.

ELECTRICAL CHARACTERISTICS (T_a = 25 °C)

SYMBOLS	PARAMETERS/TEST CONDITIONS		MIN	TYP	MAX	UNIT
NF	Noise Figure, f=12GHz Vds=2V, Ids=15mA	**EPB018A5-70**		0.50	0.60	dB
		EPB018A7-70		0.65	0.80	
		EPB018A9-70		0.95	1.20	
Ga	Associated Gain, f=12GHz Vds=2V, Ids=15mA	**EPB018A5-70**	11.5	13.0		dB
		EPB018A7-70	11.0	12.5		
		EPB018A9-70	10.5	11.5		
P_{1dB}	Output Power at 1dB Compression Vds=3V, Ids=25mA	f=12GHz f=18GHz		15.0 15.0		dBm
G_{1dB}	Gain at 1dB Compression Vds=3V, Ids=25mA	f=12GHz f=18GHz		14.0 11.5		dB
Idss	Saturated Drain Current Vds=2V, Vgs=0V		15	45	80	mA
Gm	Transconductance Vds=2V, Vgs=0V		50	90		mS
Vp	Pinch-off Voltage Vds=2V, Ids=1.0mA			-0.8	-2.5	V
BVgd	Drain Breakdown Voltage Igd=10uA		-3	-6		V
BVgs	Source Breakdown Voltage Igs=10uA		-3	-6		V
Rth	Thermal Resistance			480*		°C/W

***Overall Rth depands on case mounting.**

MAXIMUM RATINGS AT 25°C

SYMBOLS	PARAMETERS	ABSOLUTE[1]	CONTINUOUS[2]
Vds	Drain-Source Voltage	5V	4V
Vgs	Gate-Source Voltage	-3V	-2V
Ids	Drain Current	Idss	60mA
Igsf	Forward Gate Current	2mA	0.3mA
Pin	Input Power	12dBm	@1dB Compression
Tch	Channel Temperature	175°C	150°C
Tstg	Storage Temperature	-65/175°C	-65/150°C
Pt	Total Power Dissipation	285mW	240mW

Note: 1. Exceeding any of the above ratings may result in permanent damage.
2. Exceeding any of the above ratings may reduce MTTF below design goals.

Excelics Semiconductor, Inc., 2908 Scott Blvd., Santa Clara, CA 95054
Phone: (408) 970-8664 Fax: (408) 970-8998 Web Site: www.excelics.com

EPB018A5/A7/A9-70

DATA SHEET

Super Low Noise High Gain Heterojunction FET

EPB018A5-70
S-PARAMETERS
2V, 15mA

FREQ	--- S11 ---		--- S21 ---		--- S12 ---		--- S22 ---	
(GHz)	MAG	ANG	MAG	ANG	MAG	ANG	MAG	ANG
1.0	0.983	-18.6	6.245	162.2	0.019	78.9	0.530	-13.5
2.0	0.944	-37.5	5.964	144.3	0.036	65.2	0.507	-28.8
3.0	0.896	-55.5	5.582	127.7	0.050	53.6	0.485	-42.6
4.0	0.849	-72.6	5.327	112.4	0.063	43.6	0.464	-54.2
5.0	0.797	-89.2	5.111	97.6	0.074	33.1	0.421	-65.4
6.0	0.747	-103.7	4.799	83.4	0.081	23.4	0.370	-78.6
7.0	0.691	-118.6	4.503	69.9	0.085	13.9	0.344	-90.7
8.0	0.642	-132.8	4.277	57.0	0.088	4.7	0.303	-100.7
9.0	0.600	-155.6	4.189	42.7	0.093	-5.1	0.271	-111.2
10.0	0.567	-178.3	4.012	27.8	0.096	-16.3	0.228	-126.9
11.0	0.534	170.3	3.846	15.5	0.094	-26.5	0.193	-145.5
12.0	0.515	155.6	3.758	2.9	0.093	-33.1	0.177	-161.2
13.0	0.555	128.7	3.569	-12.5	0.091	-44.2	0.137	176.3
14.0	0.596	106.0	3.317	-27.1	0.088	-55.6	0.114	151.4
15.0	0.592	91.3	3.214	-41.3	0.087	-66.9	0.141	123.9
16.0	0.597	74.3	3.086	-56.8	0.083	-81.1	0.158	94.5
17.0	0.619	59.2	2.756	-69.5	0.071	-90.3	0.134	68.1
18.0	0.670	49.9	2.668	-79.4	0.071	-97.3	0.136	64.0
19.0	0.668	33.0	2.623	-95.4	0.069	-115.9	0.169	51.0
20.0	0.708	17.3	2.551	-111.1	0.064	-131.4	0.172	37.8
21.0	0.757	8.2	2.447	-125.1	0.061	-144.1	0.159	18.7
22.0	0.743	-2.5	2.325	-139.4	0.063	-159.2	0.135	14.7
23.0	0.726	-21.1	2.224	-158.5	0.065	179.4	0.115	-1.3
24.0	0.747	-39.6	2.063	-178.1	0.067	158.8	0.102	-39.6
25.0	0.709	-52.6	2.024	167.9	0.072	144.7	0.136	-56.6
26.0	0.683	-70.6	2.006	150.2	0.083	132.8	0.117	-71.3

EPB018A7-70
S-PARAMETERS
2V, 15mA

FREQ	--- S11 ---		--- S21 ---		--- S12 ---		--- S22 ---	
(GHz)	MAG	ANG	MAG	ANG	MAG	ANG	MAG	ANG
1.0	0.985	-18.9	5.754	162.0	0.021	77.1	0.677	-13.7
2.0	0.949	-38.2	5.495	143.9	0.040	63.1	0.650	-28.9
3.0	0.903	-56.2	5.137	127.2	0.055	50.5	0.622	-42.7
4.0	0.860	-73.6	4.914	111.8	0.067	39.1	0.595	-54.1
5.0	0.812	-90.4	4.726	96.9	0.079	28.5	0.549	-65.4
6.0	0.765	-104.9	4.461	82.4	0.086	17.8	0.495	-78.6
7.0	0.713	-119.9	4.189	68.6	0.092	7.3	0.464	-90.5
8.0	0.664	-134.3	3.982	55.4	0.093	-3.6	0.411	-100.6
9.0	0.621	-157.1	3.908	40.9	0.096	-12.9	0.374	-108.6
10.0	0.591	-179.4	3.759	25.7	0.098	-24.5	0.328	-121.7
11.0	0.564	169.0	3.644	12.8	0.099	-33.4	0.295	-140.0
12.0	0.541	153.2	3.551	-0.8	0.098	-43.3	0.266	-157.6
13.0	0.574	126.2	3.360	-16.6	0.096	-54.9	0.210	-174.2
14.0	0.609	103.6	3.093	-31.7	0.090	-66.7	0.173	167.6
15.0	0.598	88.8	2.985	-46.4	0.090	-78.4	0.187	139.8
16.0	0.597	71.4	2.857	-62.2	0.085	-92.9	0.194	109.8
17.0	0.612	55.7	2.548	-75.5	0.072	-102.8	0.155	89.8
18.0	0.661	46.6	2.472	-85.8	0.076	-105.2	0.183	89.7
19.0	0.657	29.0	2.381	-102.1	0.076	-126.2	0.221	68.8
20.0	0.697	13.2	2.286	-118.1	0.071	-141.6	0.240	56.1
21.0	0.740	4.4	2.173	-131.8	0.068	-155.3	0.221	40.9
22.0	0.728	-5.8	2.067	-145.9	0.070	-167.9	0.210	36.8
23.0	0.717	-24.4	1.958	-164.5	0.071	172.5	0.188	21.8
24.0	0.743	-41.8	1.807	176.3	0.071	151.8	0.154	-5.5
25.0	0.710	-53.5	1.757	161.7	0.075	138.3	0.174	-28.1
26.0	0.689	-69.1	1.759	145.4	0.084	124.1	0.152	-47.5

EPB018A7-70
Noise Parameters
Vds=2V, Ids=15mA

Freq.	Gamma Opt		Nfmin	
(GHz)	(MAG)	(ANG)	(dB)	Rn/50
2	0.76	25	0.37	0.26
4	0.65	56	0.43	0.22
6	0.51	84	0.48	0.16
8	0.41	118	0.55	0.11
10	0.26	159	0.61	0.08
12	0.26	-144	0.68	0.08
14	0.32	-82	0.89	0.18
16	0.40	-46	1.10	0.29
18	0.40	-26	1.30	0.45
20	0.51	8	1.45	0.55
22	0.41	27	1.69	0.61
24	0.48	75	1.83	0.59
26	0.52	108	2.05	0.40

Apéndice D
Datasheet del amplificador MAX2430

19-1093; Rev 2; 1/98

EVALUATION KIT MANUAL FOLLOWS DATA SHEET

Low-Voltage, Silicon RF Power Amplifier/Predriver

MAX2430

General Description

The MAX2430 is a versatile, silicon RF power amplifier that operates directly from a 3V to 5.5V supply, making it suitable for 3-cell NiCd or 1-cell lithium-ion battery applications. It is designed for use in the 800MHz to 1000MHz frequency range and, at 915MHz, can produce +21dBm (125mW) of output power with greater than 32dB of gain at V_{CC} = 3.6V.

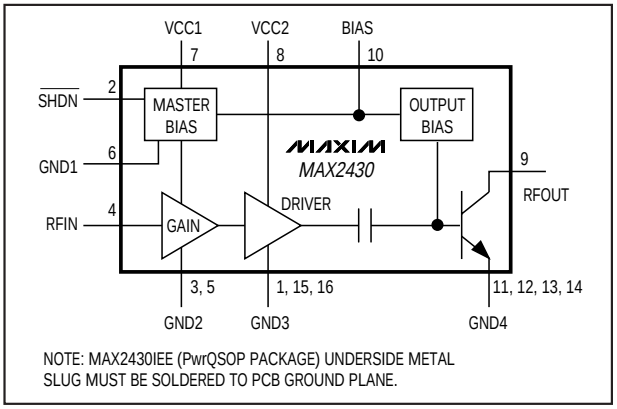
A unique shutdown function provides an off supply current of typically less than 1µA to save power during "idle slots" in time-division multiple-access (TDMA) transmissions. An external capacitor sets the RF output power envelope ramp time. External power control is also possible over a 15dB range. The amplifier's input is matched on-chip to 50Ω. The output is an open collector that is easily matched to a 50Ω load with few external components.

The MAX2430 is ideal as a driver amplifier for portable and mobile telephone systems, or as a complete power amplifier for other low-cost applications, such as those in the 915MHz spread-spectrum ISM band. It is fabricated with Maxim's high-frequency bipolar transistor process and is available in a thermally enhanced, 16-pin narrow SO and miniature 16-pin PwrQSOP packages with heat slug.

Applications

Digital Cordless Phones
915MHz ISM-Band Applications
Two-Way Pagers
Wireless LANs
Cellular Phones
AM and FM Analog Transmitters

Functional Diagram



NOTE: MAX2430IEE (PwrQSOP PACKAGE) UNDERSIDE METAL SLUG MUST BE SOLDERED TO PCB GROUND PLANE.

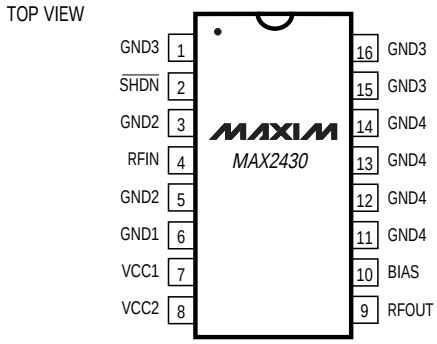
Features

- **Operates Over the 800MHz to 1000MHz Frequency Range**
- **Delivers 125mW at 915MHz from +3.6V Supply (100mW typical from +3.0V supply)**
- **Operates Directly from 3-Cell NiCd or 1-Cell Lithium-Ion Battery**
- **Over 32dB Power Gain**
- **RF Power Envelope Ramping is Programmable with One External Capacitor**
- **Input Matched to 50Ω (VSWR < 2:1)**
- **15dB Output Power Control Range**
- **1µA Typical Shutdown Current**

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX2430IEE	-20°C to +85°C	16 PwrQSOP
MAX2430ISE	-20°C to +85°C	16 Narrow SO

Pin Configuration



Narrow SO/PwrQSOP

Low-Voltage, Silicon RF Power Amplifier/Predriver

ABSOLUTE MAXIMUM RATINGS

VCC1, VCC2 .....+6V
$\overline{SHDN}$, BIAS.....-0.3V, (V_{CC} + 0.3V)
RFIN.....-0.3V, +2V
P_{RFIN}.....-3dBm
Continuous Power Dissipation (T_A = +70°C)
PwrQSOP (derate 20mW/°C above +70°C).....1.6W
Narrow SO (derate 20mW/°C above +70°C) 1.6W

Operating Temperature Range .....-20°C to +85°C
Storage Temperature Range .....-65°C to +160°C
Lead Temperature (soldering, 10sec) .....+300°C

Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated in the operational sections of the specifications is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = VCC1 = VCC2 = RFOUT = 3V to 5.5V, GND1 = GND2 = GND3 = GND4 = 0V, $\overline{SHDN}$ = 2.2V, BIAS = open, RFIN = open, T_A = -20°C to +85°C, unless otherwise noted.)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Supply Voltage Range	V_{CC}		3		5.5	V
Supply Current	I_{CC}	No RF input applied, V_{CC} = 5.5V		52	70	mA
Shutdown Supply Current	$I_{CC(OFF)}$	$\overline{SHDN}$ = low		1	10	µA
BIAS Pin Voltage	V_{BIAS}	BIAS pin open		2.2		V
$\overline{SHDN}$ High Input	$V_{SHDN(HI)}$		2.2		V_{CC}	V
$\overline{SHDN}$ Low Input	$V_{SHDN(LO)}$				0.4	V
$\overline{SHDN}$ Bias Current	I_{SHDN}	$\overline{SHDN}$ = V_{CC}			18	µA

AC ELECTRICAL CHARACTERISTICS

(MAX2430 EV kit, f = 915MHz, V_{CC} = 3.6V, $\overline{SHDN}$ = V_{CC}, RFOUT matched to 50Ω resistive load, output measurements taken after matching network, T_A = +25°C, unless otherwise noted.) (Note 1)

PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Frequency Range		(Note 2)		800		1000	MHz
P_{OUT} at 1dB Compression	P_{1dB}	V_{CC} = 3.6V		20	21.4		dBm
		V_{CC} = 3.0V		19	20.4		
Power Gain	G_P	P_{RFIN} = -20dBm	MAX2430ISE	32	34		dB
			MAX2430IEE	31	33		
Output IM3	OIM3	f1 = 915MHz, f2 = 916MHz, P_{OUT} per tone = 14dBm			-30		dBc
2nd Harmonic		P_{OUT} = P_{1dB}			-26		dBc
3rd Harmonic		P_{OUT} = P_{1dB}			-40		dBc
Efficiency	η	P_{OUT} = P_{1dB}			24		%
Supply Current	I_{CCRF}	P_{OUT} = P_{1dB}			160		mA
Maximum Input VSWR	$VSWR_{IN}$	RFIN connected to 50Ω source			2:1		
Maximum Output Load Mismatch	$VSWR_{OUT}$	V_{CC} = 3V to 5.5V, P_{RFIN} ≤ -10dBm (Note 3)			8:1		
Maximum Output Load Mismatch for Stability	$VSWR_{OUT}$	V_{CC} = 3V to 5.5V, P_{RFIN} ≤ -12dBm (Note 4)			6:1		
Noise Figure	NF				7		dB

AC ELECTRICAL CHARACTERISTICS (continued)

(MAX2430 EV kit, f = 915MHz, V_{CC} = 3.6V, $\overline{SHDN}$ = V_{CC}, output matched to 50Ω resistive load, output measurements taken after matching network, T_A = +25°C, unless otherwise noted.) (Note 1)

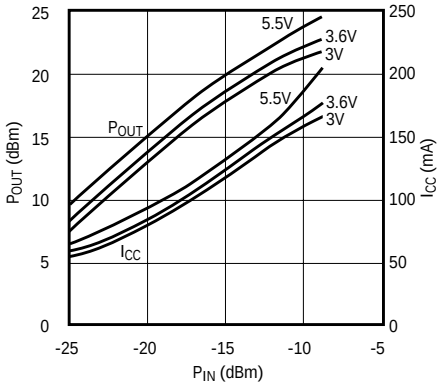
PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
RFIN to RFOUT Isolation		$\overline{SHDN}$ = 0.4V, P_{IN} = -10dBm	MAX2430ISE		50		dB
			MAX2430IEE		47		dB
Turn-On/Off Times		BIAS pin capacitor C1 = 120pF			1		µs
		BIAS pin capacitor C1 = 2.2nF			10		

Note 1: Minimum and maximum parameters are guaranteed by design.
Note 2: For optimum performance at a given frequency, output matching network must be designed for maximum output power. See *Applications Information* section. Operation outside this frequency range is possible but has not been characterized.
Note 3: No damage to the device.
Note 4: All non-harmonically related outputs are more than 60dB below the desired signal for any electrical phase.

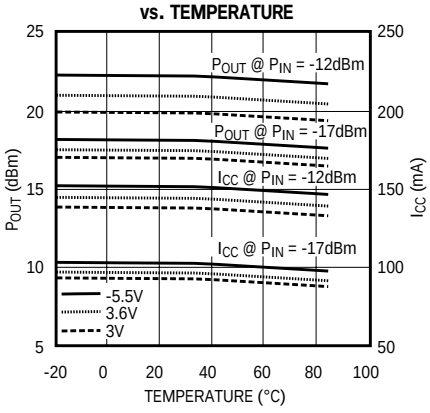
Typical Operating Characteristics

(MAX2430EVKIT-SO, f = 915MHz, V_{CC} = 3.6V, $\overline{SHDN}$ = V_{CC}, output matched to 50Ω resistive load, output measurements taken after matching network, T_A = +25°C, unless otherwise noted.)

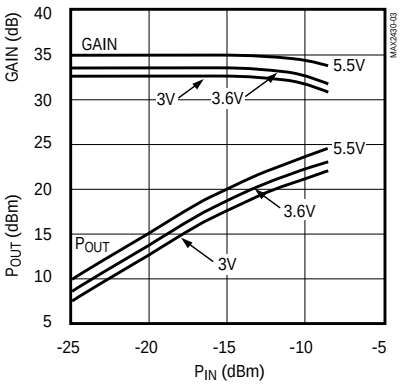
OUTPUT POWER AND CURRENT vs. INPUT POWER



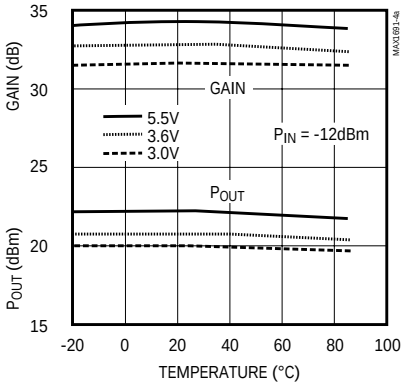
OUTPUT POWER AND CURRENT vs. TEMPERATURE



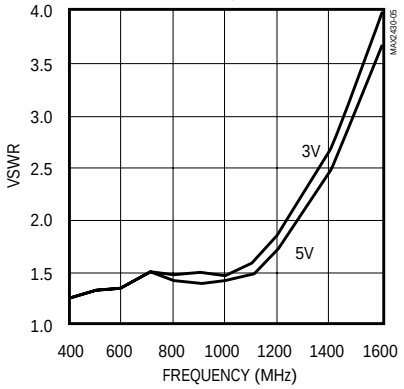
OUTPUT POWER AND GAIN vs. INPUT POWER



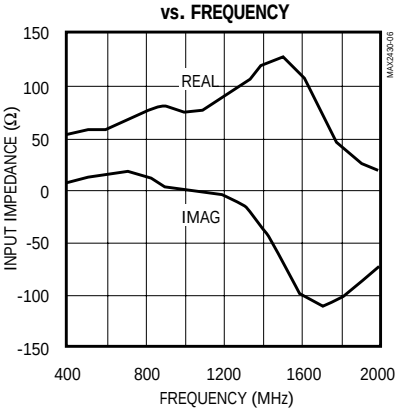
OUTPUT POWER AND GAIN vs. TEMPERATURE (NORMAL OPERATING MODE)



INPUT VSWR vs. FREQUENCY



RF INPUT IMPEDANCE vs. FREQUENCY

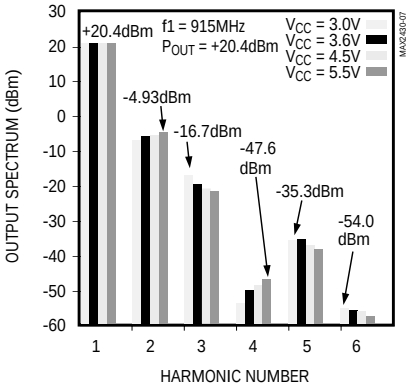


Low-Voltage, Silicon RF Power Amplifier/Predriver

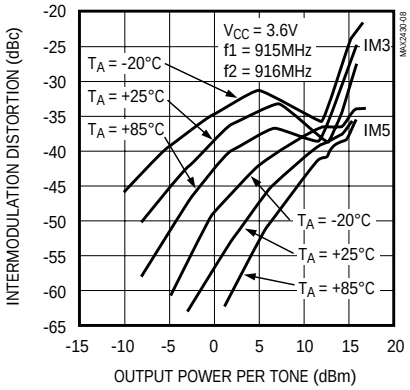
Typical Operating Characteristics (continued)

(MAX2430EVKIT-SO, f = 915MHz, V_{CC} = 3.6V, $\overline{SHDN}$ = V_{CC}, output matched to 50Ω resistive load, output measurements taken after matching network, T_A = +25°C, unless otherwise noted.)

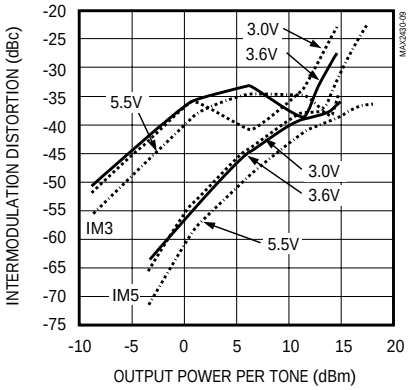
OUTPUT POWER AND HARMONICS



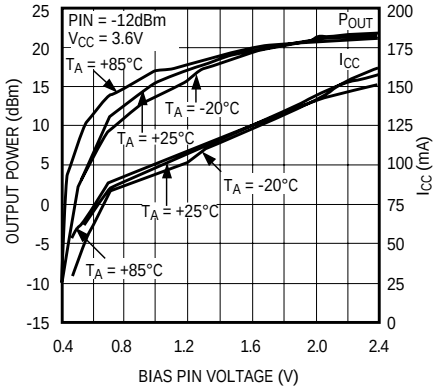
INTERMODULATION DISTORTION vs. OUTPUT POWER AND TEMPERATURE



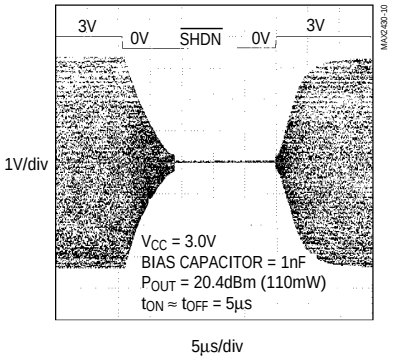
INTERMODULATION DISTORTION vs. OUTPUT POWER AND V_{CC}



OUTPUT POWER AND SUPPLY CURRENT vs. EXTERNAL CONTROL VOLTAGE



RF OUTPUT ENVELOPE CHARACTERISTICS vs. SHUTDOWN CONTROL



Pin Description

PIN	NAME	FUNCTION
1, 15, 16	GND3	Driver Stage Ground. Connect directly to ground plane.
2	$\overline{SHDN}$	Shutdown Input (TTL/CMOS)
3, 5	GND2	Input Stage Ground. Connect directly to ground plane.
4	RFIN	RF Input. Internally matched to 50Ω. Requires series DC-blocking capacitor.
6	GND1	Bias Circuitry Ground. Connect directly to ground plane.
7	VCC1	Bias Circuitry Supply. Connect to supply. Bypass with 1000pF capacitor.
8	VCC2	Driver Stage Output. Connect to supply through inductor (see *Applications Information*).
9	RFOUT	Output Transistor. Open Collector.
10	BIAS	Output Stage Bias Pin. Connect capacitor to GND to control start-up power envelope. Drive directly for power control (see *Applications Information*).
11–14	GND4	Output Stage Ground. Connect directly to ground plane.

Note: *MAX2430IEE (PwrQSOP package) underside metal slug must be soldered to PCB ground plane.*

Detailed Description

The MAX2430 consists of a large power output transistor driven by a capacitively coupled driver stage (see *Functional Diagram*). The driver and front-end gain stages are DC-connected and biased on-chip from the master bias cell. The master bias cell also controls the output stage bias circuit. The input impedance at the RFIN pin is internally matched to 50Ω, while the output stage must be tuned and filtered externally for any narrow-band frequency range of interest between 800MHz and 1000MHz.

The driver amplifier requires an external inductor at the VCC2 pin to provide DC bias and proper matching to the output stage. This inductor's value depends on the package type and frequency range of operation; typically it will vary between 5nH and 22nH.

The output transistor at the RFOUT pin requires an external RF choke inductor connected to the supply for DC bias, and a matching network to transform the desired external load impedance to the optimal internal load impedance of approximately 15Ω.

The MAX2430 includes a unique shutdown feature. The TTL/CMOS-compatible $\overline{SHDN}$ input allows the device to be shut down completely without the use of any external components. Also, the RF output power envelope ramp time can be programmed with a single external capacitor connected between the BIAS pin and ground. Pulling the shutdown pin ($\overline{SHDN}$) high powers on the master bias circuit, which in turn charges the external capacitor tied to the BIAS pin using a controlled current. The voltage at BIAS controls the output power level, which ramps until the BIAS pin is internally clamped to approximately 2.2V. The envelope rampdown time is controlled in a similar manner when the $\overline{SHDN}$ pin is pulled low.

Variable output power control over a 15dB range is also possible by forcing the voltage on the BIAS pin externally from 0.6V to 2.4V.

During the on state ($\overline{SHDN}$ = high), the power-supply bias current is typically 52mA with no RF applied to the input. During the off state ($\overline{SHDN}$ = low), the supply current is typically reduced to less than 1µA.

Low-Voltage, Silicon RF Power Amplifier/Predriver

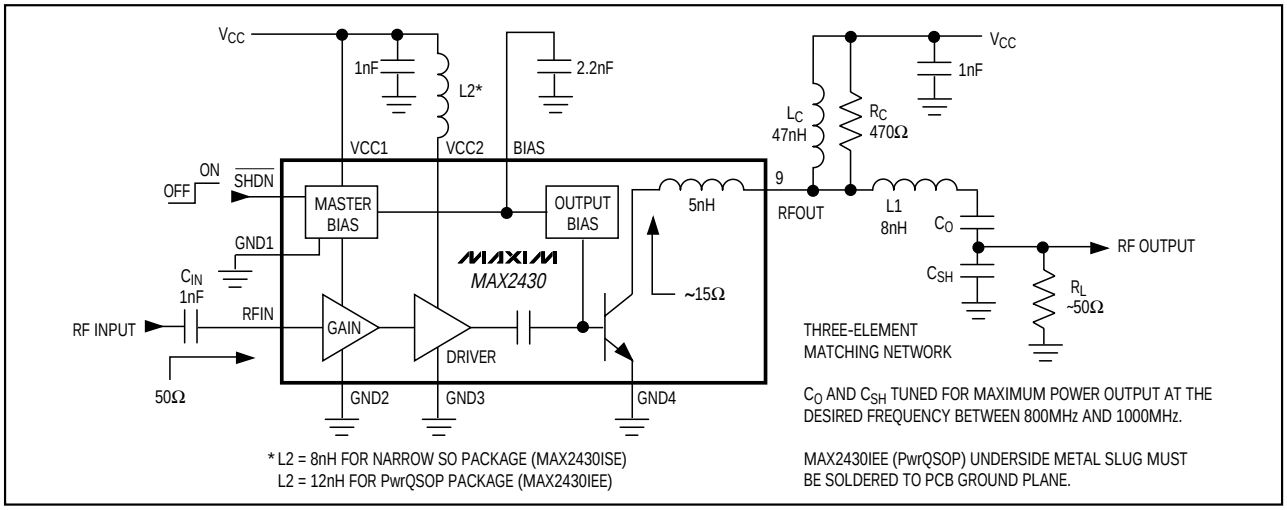


Figure 1. Typical Application Circuit

Applications Information

Output Matching

The optimum internal load impedance seen by RFOUT is approximately 15Ω. This on-chip low drive impedance provides maximum power transfer and best efficiency under low (3V) supply conditions where the voltage-swing headroom is limited. For example, driving an output power of 21.3dBm (135mW) into 50Ω translates to a 7.35Vp-p swing at the output. An RF amplifier would require at least a 4.5V supply to drive a 50Ω load directly. However, driving 21.3dBm into 15Ω translates to 4.02Vp-p. The MAX2430 can achieve a voltage swing of 4.02Vp-p or 2.01Vp from a 3V supply voltage without saturating the output transistor.

Figure 1 shows the MAX2430 configured for 800MHz to 1000MHz operation. The output matching circuitry converts the desired 50Ω load impedance to the 15Ω optimal load seen by the output transistor's collector. This configuration uses a low-loss, controlled-Q inductor network. Starting from the RFOUT pin, this network consists of a series L (which includes the 5nH package parasitic inductance), series C, and shunt C. The design equations for this network are as follows:

R1 = Output resistance as seen by the collector ~15Ω

R_L = Desired load resistance

The controlled-Q inductor network requires that $R_L > R1$ and $Q > \sqrt{(R_L / R1 - 1)}$. Choose Q and compute matching components as given below:

$$\text{Let} \quad A = \sqrt{(R_L \times R1 - R1^2)}$$

$$X_L = Q \times R1$$

$$X_{Co} = X_L - A$$

$$X_{Csh} = R_L \times R1 / A$$

$$L1 = X_L / \omega - 5\text{nH of package inductance}$$

$$C_O = \frac{1}{\omega X_{Co}}$$

$$C_{SH} = \frac{1}{\omega X_{Csh}}$$

where ω equals the center frequency in radians/second.

Recommended starting values for L1 and L2 are given in Table 1.

Table 1. Recommended L1 and L2 Starting Values

f = ω / 2π (MHz)	L1(nH)	MAX2430ISE L2(nH)	MAX2430IEE L2(nH)
400 to 600*	22	12	18
600 to 800*	15	8	12
800 to 1000	8	8	12

*Not characterized

An overall loaded Q ≤ 5 can be achieved with readily available surface-mount components. This network absorbs the parasitic elements of the surface-mount components in such a way that they do not negatively impact the stopband characteristics; in fact, they can improve the overall stopband attenuation with properly chosen components. High-Q components (Q > 100) that have self-resonance near the 3rd harmonic of the intended output frequency should provide good passband characteristics with low loss, while offering good attenuation of the undesired 2nd and 3rd harmonics that are generated. Note that most applications will require extra filtering components and good shielding after the matching network to ensure absolute attenuation of out-of-band signals in order to meet out-of-band spurious suppression requirements.

Output Mismatch Considerations

The MAX2430 will typically withstand an output load mismatch of VSWR = 6:1 at any electrical phase without exhibiting oscillatory behavior over the entire supply voltage range of 3V to 5.5V. Resistor R_C enhances stability under load mismatch conditions and does not affect normal operation of the circuit.

BIAS Pin

The voltage at the BIAS pin controls the output power transistor biasing. At BIAS = 0.6V, the output transistor is biased to Class C, resulting in low gain and relatively nonlinear power. Above 2V, the output stage is biased to Class AB. Note that changing the bias voltage may degrade the output transistor's stability.

The shutdown pin ($\overline{SHDN}$) controls the master bias circuit, which in turn provides a control current of approximately ±500µA to the external capacitor connected to the BIAS pin. When $\overline{SHDN}$ transitions from low to high, the BIAS pin capacitor charges up and clamps at approximately 2.2V. When $\overline{SHDN}$ transitions from high to low, the BIAS pin capacitor is discharged to nearly ground. This results in a power-up/power-down ramping of the RF envelope, which can be approximated by the following equation:

$$t_{ramp} \cong C_{BIAS} \times 2.2V / 0.5mA = 4400\Omega \times C_{BIAS}$$

Therefore, a 2.2nF capacitor will give approximately 10µs ramp time.

The BIAS pin can also be used to control the final output power and gain over a 15dB range, by forcing the BIAS pin voltage externally between 0.6V and 2.4V. Note that the BIAS pin driver must be able to source/sink 700µA. Forcing the BIAS pin directly in this manner disrupts the RF envelope timing function. To avoid this, place a diode in series with the BIAS pin control circuit, as shown in Figure 2.

Note that when using the BIAS pin for power control, linearity is much degraded at the lower power levels.

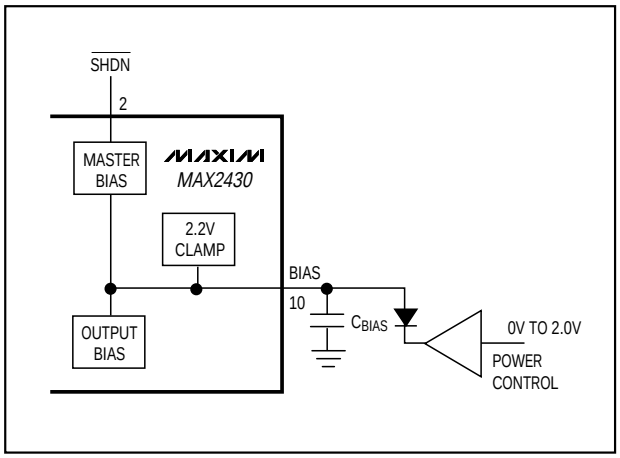


Figure 2. Power-Control Application Using BIAS Pin

Operating Frequency Range

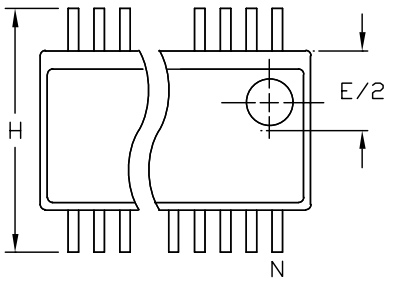
The MAX2430 has been characterized for operation in the 800MHz to 1000MHz range. Operation outside this range is possible, but the following issues must be considered:

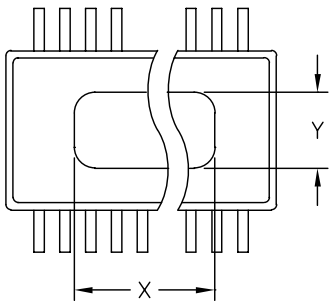
- Gain increases substantially at lower frequencies, possibly causing stability problems.
- Useful gain and output power levels drop rapidly above 1000MHz.

Low-Voltage, Silicon RF Power Amplifier/Predriver

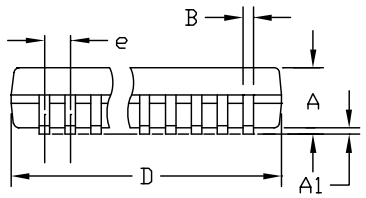
Package Information

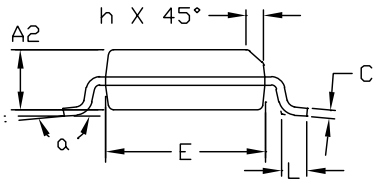
PSSOPPS.EPS





DIM	INCHES MIN	INCHES MAX	MILLIMETERS MIN	MILLIMETERS MAX
A	.061	.068	1.55	1.73
A1	.004	.010	0.127	0.25
A2	.055	.059	1.40	1.55
B	.008	.012	0.20	0.31
C	.007	.010	0.19	0.25
D	SEE VARIATIONS			
E	.150	.157	3.81	3.99
e	.025 BSC		0.635 BSC	
H	.230	.244	5.84	6.20
h	.010	.016	0.25	0.41
L	.016	.035	0.41	0.89
N	SEE VARIATIONS			
X	SEE VARIATIONS			
Y	.071	.087	1.803	2.209
α	0°	8°	0°	8°





VARIATIONS:

	INCHES MIN.	INCHES MAX.	MILLIMETERS MIN.	MILLIMETERS MAX.	N	
D	.189	.196	4.80	4.98	16	AA
X	.107	.123	2.717	3.124		

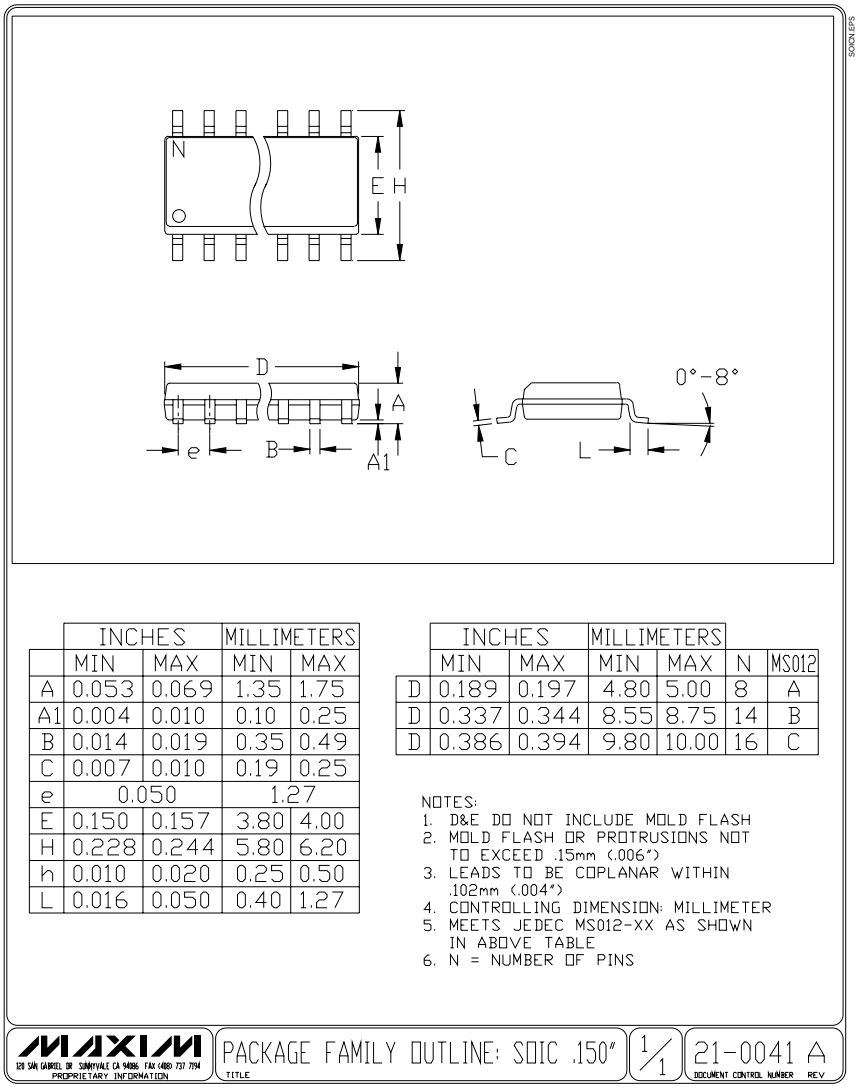
D	.386	.391	9.80	9.98	28	AB
X	.271	.287	6.883	7.290		

NOTES:
1. D & E DO NOT INCLUDE MOLD FLASH OR PROTRUSIONS
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED .006"
3. CONTROLLING DIMENSIONS: INCHES

MAXIM
PROPRIETARY INFORMATION
TITLE:
PACKAGE OUTLINE, POWER QSOP (PSSOP2)

APPROVAL	DOCUMENT CONTROL NO.	REV	
	21-0063	A	1/1

Package Information (continued)



	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.053	0.069	1.35	1.75
A1	0.004	0.010	0.10	0.25
B	0.014	0.019	0.35	0.49
C	0.007	0.010	0.19	0.25
e	0.050		1.27	
E	0.150	0.157	3.80	4.00
H	0.228	0.244	5.80	6.20
h	0.010	0.020	0.25	0.50
L	0.016	0.050	0.40	1.27

	INCHES		MILLIMETERS			
	MIN	MAX	MIN	MAX	N	MS012
D	0.189	0.197	4.80	5.00	8	A
D	0.337	0.344	8.55	8.75	14	B
D	0.386	0.394	9.80	10.00	16	C

NOTES:
1. D&E DO NOT INCLUDE MOLD FLASH
2. MOLD FLASH OR PROTRUSIONS NOT TO EXCEED .15mm (.006")
3. LEADS TO BE COPLANAR WITHIN .102mm (.004")
4. CONTROLLING DIMENSION: MILLIMETER
5. MEETS JEDEC MS012-XX AS SHOWN IN ABOVE TABLE
6. N = NUMBER OF PINS

MAXIM 120 SAN GABRIEL DR SUNNYVALE CA 94086 FAX (408) 737 7194 PROPRIETARY INFORMATION

PACKAGE FAMILY OUTLINE: SOIC .150" TITLE

1/1

21-0041 A DOCUMENT CONTROL NUMBER REV.

NOTES

Índice de Figuras

Índice de Tablas

Índice de Códigos

Bibliografía

[1] Eduardo Amores, *Control de instrumentos mediante el bus GPIB programado con Matlab*, Master's thesis, Universidad Autónoma de Barcelona (UAB), July 2010.

[2] Eugenia Bahit, *Curso: Python para principiantes*, *http://librosweb.es/libro/python/*, 2012.

[3] María España Borrero, *Herramienta Software para el Control Remoto de una Fuente de Alimentación mediante una Interfaz Gráfica*, Master's thesis, Universidad de Sevilla, July 2011.

[4] Rafael Chacón, *La instrumentación virtual en la enseñanza de la Ingeniería Electrónica*, Acción Pedagógica, Universidad de los Andes (Venezuela) **11** (2002), no. 1, 80–89.

[5] LXI Consortium, *Página Web Oficial de LAN eXtensions for Instrumentation*, Consulta: mayo 2018.

[6] Departamento de Electrónica, *Apuntes de Mediciones Electrónicas*, Universidad Nacional de Mar del Plata.

[7] Departamento de Ingeniería Electrónica, *Apuntes de Laboratorio de Sistemas Electrónicos Digitales*, Universidad Politécnica de Valencia.

[8] Quentin Docter, Emmett Dulaney, and Toby Skandier, *CompTIA A+ Complete Deluxe Study Guide Recommended Courseware: Exams 220-801 and 220-802*, Sybex, 2012.

[9] Jeffrey Elkner, Allen B. Downey, and Chris Meyers, *How to Think Like a Computer Scientist: Learning with Python*, Artpower International, 2016.

[10] Néstor Gabriel Forero, *Normas de Comunicación en Serie: RS-232, RS-422 y RS-485*, Revista Ingenio Libre **11** (2012), no. 13, 86–94.

[11] IVI Foundation, *VISA Specification*, *http://www.ivifoundation.org/specifications/default.aspx*, Consulta: abril 2018.

[12] José María Herrera and Luis Miguel Sánchez, *Apuntes de Óptica*, ch. 4, Universidad Complutense de Madrid, 2013.

[13] Nationals Instruments, *Página Web Oficial de Nationals Instruments*, *http://www.ni.com/es-es.html*, Consulta: mayo 2018.

[14] Kim Otte, *What is PXI? Your Questions Answered*, *https://blog.pickeringtest.com/bid/179876/what-is-pxi-your-questions-answered*, June 2013.

[15] Iván Pulido, *Control remoto de instrumentación para radiocomunicación mediante Python*, Master's thesis, Universidad de Sevilla, 2017.

[16] PyVISA, *PyVISA: Control your instruments with Python*, *https://pyvisa.readthedocs.io/en/stable/*, Consulta: junio 2018.

[17] Juan Ranchal, *PCI Express, guía para entender el bus y sus variantes*, *https://www.muycomputer.com/2017/11/09/pci-express-guia/*, November 2017.

[18] Rohde & Schwarz, *Página Web Oficial de Rohde & Schwarz*, https://www.rohde-schwarz.com/, Consulta: mayo 2018.

[19] Bartolomé Sintes, *Curso de iniciación a la programación en Python*, http://www.mclibre.org/consultar/python/, 2018.

[20] Keysight Technologies, *6620A Series Multiple-Output, 40-105 W DC Power Supplies, GPIB*.

[21] Keysight Technologies, *Operating Manual Multiple-Output Linear System DC Power Supply Agilent Models 6625A, 6626A, 6628A, and 6629A*.

[22] Carlos Villagómez, *USB*, http://es.ccm.net/contents/407-usb-bus-de-serie-universal, December 2017.

[23] Carlos Vázquez, *Puertos PCI Express: ¿Qué son y para qué sirven?*, http://www.islabit.com/76371/puertos-pci-express-tamanos.html, October 2017.

[24] Rosita Wachenchauzer, Margarita Manterola, Maximiliano Curia, Marcos Medrano, and Nicolás Paez, *Algoritmos dePrograma ción con Python*, http://librosweb.es/libro/algoritmos_python/, March 2011.

[25] Dr. Pápay Zsolt, *GPIB Tutorial*, Universidad de Tecnología y Economía de Budapest (BME), 2007.